MICA BOOKS
$6.95

Pocket Guide

Assembly Language for the 8085

Noel Morris

Programming Pocket Guides

Pitman Programming Pocket Guides

Programming	John Shelley
BASIC	Roger Hunt
COBOL	Ray Welland
FORTRAN	Philip Ridler
Pascal	David Watt
FORTRAN 77	Clive Page
Programming for the BBC Micro	Neil Cryer and Pat Cryer
Assembly Language for the 6502	Bob Bright
Assembly Language for the Z80	Julian Ullmann
Programming for the Apple	John Gray
The Sinclair Spectrum	Steven Vickers
The Commodore 64	Boris Allan
UNIX	Lawrence Blackburn and Marcus Taylor
LOGO	Boris Allan
Assembly Language for the 8085	Noel Morris

This series of pocket size reference guides provides you with reliable descriptions of the salient features of all the important languages and micros.

The Pocket Guide Programming is intended for those who have had no programming experience and provides you with the lead-in to the other programming language titles.

The Publishers would welcome suggestions for further improvements to this series. Please write to Alfred Waller at the address below.

PITMAN PUBLISHING LTD
128 Long Acre, London WC2E 9AN

Associated companies
Pitman Publishing Pty Ltd, Melbourne
Pitman Publishing New Zealand Ltd, Wellington
Copp Clark Pitman, Toronto

Consultant Editor: David Hatter

First edition, 1984

© Noel Morris 1984

All rights reserved.

Printed in Great Britain at The Pitman Press, Bath

ISBN 0 273 02123 0

Index

How to use this Pocket Guide

This Guide provides coverage not only of every feature of the Intel 8085 assembly language, but of the use of programmable Input/Output ports together with typical programs. The Intel 8085A microprocessor chip is identical to the 8085 chip, but uses a different manufacturing technology.

Details are given not only of the register array in the CPU, but also full information relating to the complete instruction set, operation codes, execution times, and the number of bytes needed by each instruction.

This Guide also provides a list of operating codes in numerical order together with instruction mnemonics. Subroutines and interrupt handling facilities are described, together with information about nested subroutines and multilevel interrupts. Also included is a detailed study of a programmable I/O port which contains RAM and a programmable timer. Examples are included not only in the body of the text, but also at the end of the Guide.

8085A Central Processing Unit (CPU)

The Intel 8085A CPU is housed in a 40-pin dual-in-line package whose instruction set is software compatible with that of the Intel 8080 CPU, and provides the following facilities.

246 instructions
6 addressing modes
16-bit address bus (the low eight bits are multiplexed with the data bus)
8-bit bidirectional data bus (multiplexed with the low byte of the address bus)
Crystal frequency up to 12 MHz (the resulting system clock frequency is up to 6 MHz)
Interrupt facilities—maskable and nonmaskable
Stack operation
On-chip serial input and output

The 8085A CPU includes the following: a register array (detailed in the section on the programming model of the CPU), an address buffer (for the high byte of the address), a data/address buffer (for

the multiplexed data/address bus), an instruction decoder, a timing and control section, serial I/O control, interrupt control, and an arithmetic and logic unit (ALU). The manufacturers' data sheet should be consulted for full details of the architecture of the CPU, timing diagrams, pin connections, and electrical characteristics.

The 8085 chip

The 8085 chip is shown in Fig. 1. It is housed in a 40-pin DIP and is energized by a single 5V supply; the frequency-determining

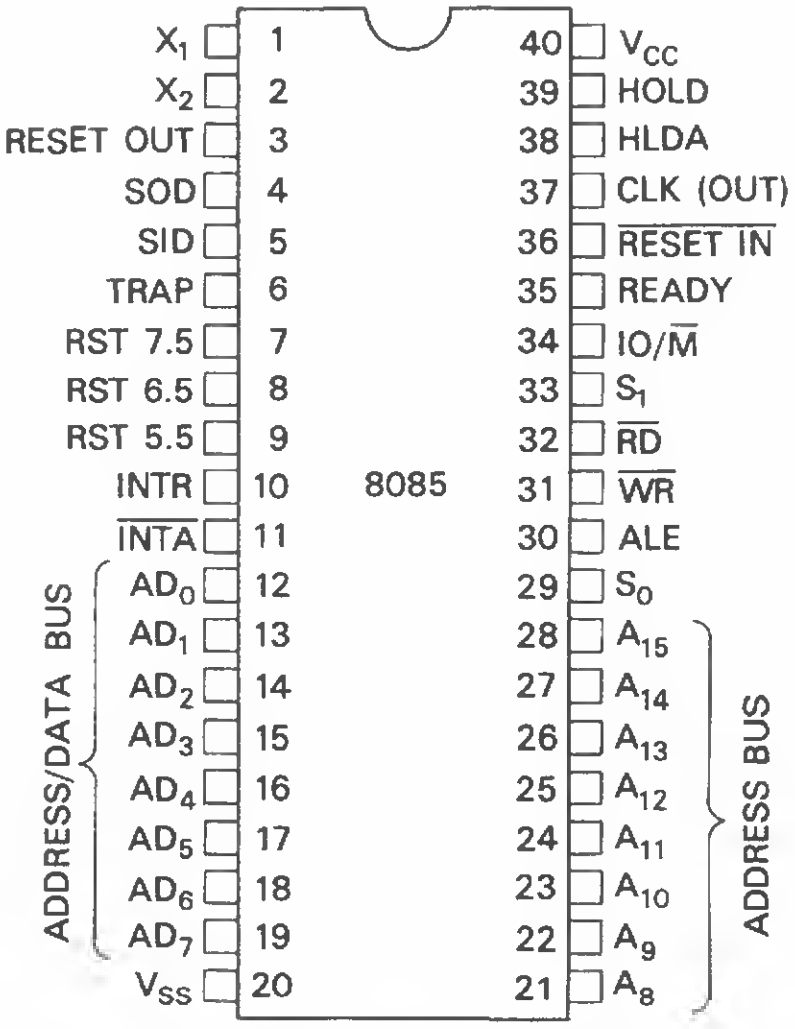
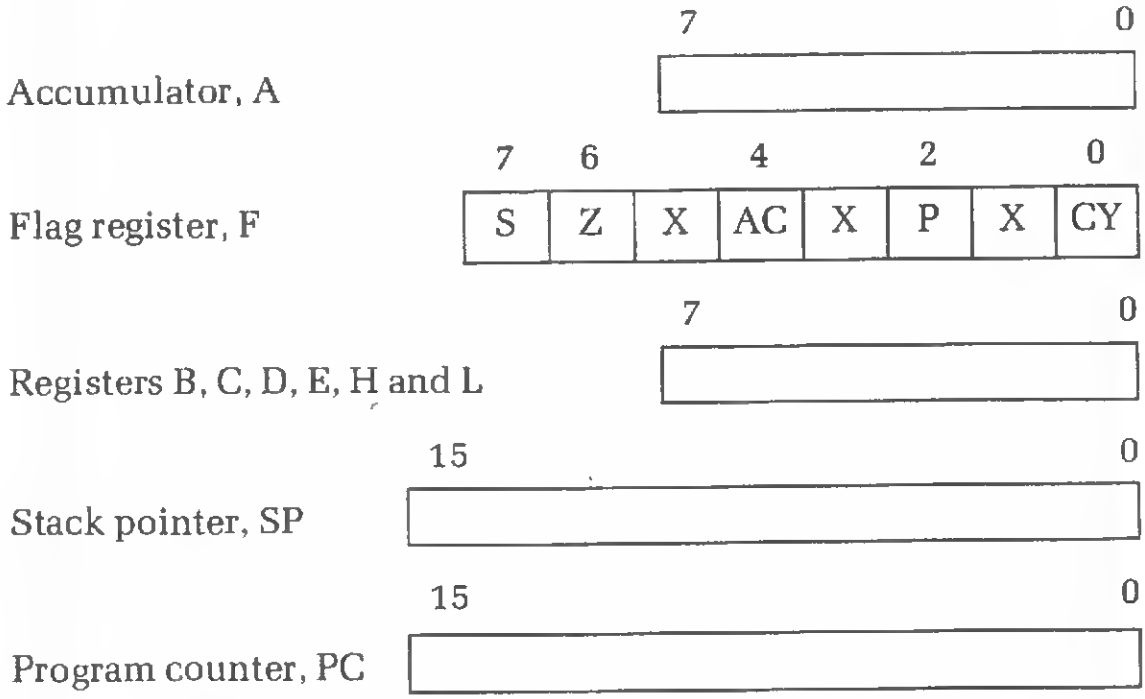


Fig. 1

element is a crystal which is connected between pins X_1 and X_2. The data bus is multiplexed as follows: the low-order eight bits of the address bus is described as the Address/Data bus (AD_0 to AD_7), this bus acting as the eight low-order bits of the Address bus (A_0 to A_7) at the time when the Address Latch Enable (ALE) line is high, and as the Data bus at other times. The high-order bits of the Address bus (A_8 to A_{15}) are output on the 'Address bus' lines.

Programming model of the CPU

The programming model of the 8085A is shown below



Accumulator, A

This is an 8-bit register used in many arithmetic and logic operations; the result of these operations remains in the accumulator.

Flag register, F

This is an 8-bit register containing eight flags (the three flags designated with an 'X' in the above diagram are not used by the instruction set of the 8085).

Bit 0 Carry flag (CY)
This flag acts as a ninth bit in certain arithmetic and logic operations, and is used to store the carry-out from bit 7 of certain operations.

Bit 1 Unused

Bit 2 Parity flag (P)
This flag checks the parity of the result of an arithmetic or logic operation (it is reset to '0' if the result has an odd number of 1's in it, and is set to '1' if the result has an even number of 1's in it).

Bit 3 Unused

Bit 4 Auxiliary carry flag (AC)
This holds the carry from bit 3 to bit 4 of the result of an arithmetic or logical operation.

Bit 5 Unused

Bit 6 Zero flag (Z)
This is set to '1' when an arithmetic or logical instruction produces a result of zero (the flag is reset to '0' when the result is nonzero).

Bit 7 Sign flag (S)
This stores the value of the sign bit (the most significant bit) of the result of an arithmetic or logical instruction.

Testing the flags

Four of the flags (CY, P, Z and S) can be tested under program control and used to initiate either a conditional jump or a conditional subroutine call, or a conditional return from a subroutine. The instructions involved are listed below.

Flag	*Jump instruction*	*Subroutine call instruction*	*Subroutine return instruction*
Z	JZ and JNZ	CZ and CNZ	RZ and RNZ
CY	JC and JNC	CC and CNC	RC and RNC
P	JPE and JPO	CPE and CPO	RPE and RPO
S	JM and JP	CM and CP	RM and RP

The conditions specified are as follows

Z	zero (Z = 1)
NZ	nonzero (Z = 0)
C	carry (CY = 1)
NC	no carry (CY = 0)
PE	parity even (P = 1)
PO	parity odd (P = 0)
M	minus (S = 1)
P	plus (S = 0)

Registers B, C, D, E, H and L

These are general-purpose 8-bit registers used in many arithmetic and logic operations. Data can be moved from any one of these registers to any other register (or to itself for that matter!).

Register pairs

Specified 8-bit registers can be combined to form a 16-bit register pair (rp). The groupings are as follows.

rp PSW (Processor Status Word) comprises the accumulator and the flag register.
rp B comprises the register pair B and C.
rp D comprises the register pair D and E.
rp M comprises the register pair H and L (this register pair is frequently used as a memory pointer, M).

The 16-bit stack pointer (SP) is also regarded as a register pair for programming purposes.

Register pair H and L

The register pair H and L (referred to as M in certain instructions) is the primary *data pointer* in the Intel 8085. It holds the 16-bit address of data being accessed by the CPU; in general, the H and L register pair should be reserved by the programmer for use as a memory pointer.

Stack pointer, SP

This is a 16-bit register which holds the absolute address of the current top of the stack the stack can be implemented anywhere in the addressable memory. For the purposes of programming, the SP is regarded as a register pair.

Program counter, PC

This 16-bit register contains the address of the next byte to be fetched from the memory.

Language elements

Machine code

A machine code instruction in the 8085 comprises either one, two or three bytes. The first byte is the *operation code* (opcode). The second and third bytes (the *operand*) of a multi-byte instruction contains either data or a 16-bit address. For example, the 3-byte machine code instruction C3 17 20 is interpreted as follows:

C3 first byte—the opcode for an unconditional JuMP instruction (mnemonic JMP)
17 second byte—low byte of an address
20 third byte—high byte of an address

The above instruction causes a programmed 'jump' to the hexadecimal address 2017.

A 1-byte machine code instruction has an opcode but no operand.

Assembly language

An assembly language instruction consists of an *operator* and an *operand*. The operator provides the same information as the opcode of the equivalent machine code instruction, and the operand provides the same information as the operand in the machine code instruction (although, in many cases, the assembly language operand provides the data in a more readily understood

manner). There is, in fact, a 1-to-1 relationship between the number of machine code instructions in a program and the number of assembly language instructions.

In some cases the operand may be absent; this occurs in an assembly language instruction which is equivalent to a 1-byte machine code instruction.

Assembly language operators

This is represented by a three-letter mnemonic which indicates the nature of the operation to be carried out.

Example:	SUB	SUBtract
	MOV	MOVe data

Assembly language operands

The operand may be expressed as a binary, a hexadecimal, an octal, a decimal, or an ASCII character as follows:

Binary	operand finishes with B, e.g., 00001111B
Hexadecimal	must start with a digit and end with an H, e.g., 1C24H or 0B5H
Octal	must end with Q or O (note: Q is less confusing), e.g., 170Q or 170O
Decimal	may end with a D (optional), e.g., 1359 or 1359D
ASCII	enclosed in quotation marks, e.g., 'THERE'

Source program

A source program is divided into four *fields* or sections, being respectively the *label field*, the *operator field*, the *operand field* and the *comment field*. A typical source program is listed below.

Label field	*Operator field*	*Operand field*	*Comment field*
	ORG	2000H	
REG15 :	MVI	C,020H	
	RST	4	
	INX	H	;INCREMENT R.P. HL
	JMP	REG05	

A colon is used after a label (except for pseudo-operations such as EQU, when a space is used). A space is left after an instruction mnemonic. A comma is used to separate operands in the operand field. A semicolon is used to separate the operand and comment fields.

The label and comment fields are optional, but the operator and operand fields must be filled (except in the case of a 1-byte instruction, when the operand field is blank).

Assembler listing

An assembler converts the source code into the object code or machine code. The output from an assembler for the program above is listed below.

Address (hex)	*Machine code*	*Label*	*Assembly language instruction*	*Comment*
2000	0E 20	REG15:	MVI C,020H	
2002	E7		RST 4	
2003	23		INX H	;INCREMENT R.P. HL
2004	C3 49 00		JMP REG05	

Addressing modes

Direct addressing or absolute addressing

Each instruction in this mode is a 3-byte instruction. The first byte is the opcode, the second byte contains the low-order byte of the address, and the third byte contains the high-order byte of the address.

Direct addressing enables any location in the 64K of addressable memory to be accessed.

Example: STA 2003H
means STore (copy) the contents of the Accumulator into the memory location having the Hexadecimal address 2003.
Assembled into machine code it becomes

32 03 20

The first byte (32) is the opcode of the STA instruction, the second byte (03) is the low byte of the destination address, and the third byte (20) is the high byte of the destination address.

Further example: JMP 20B0H
means JuMP (transfer program control) to the instruction whose opcode is in the memory location having the hexadecimal address 20B0. Assembled into machine code it becomes

C3 B0 20

where the first byte (C3) is the opcode of the JMP instruction, the second byte (B0) is the low byte of the destination address, and the third byte (20) is the high byte of the address.

Immediate addressing

These may be either a 2-byte or a 3-byte instruction. The first byte is the opcode and the second byte (or second and third bytes) contain the data.

Example of a 2-byte instruction: ADI 8FH
means ADd Immediate the contents of the accumulator to the Hexadecimal data (8F) in the second byte of the instruction, leaving the sum in the accumulator. Assembled into machine code this becomes

C6 8F

The first byte (C6) is the opcode of the ADI instruction and the second byte (8F) is the data to be added to the contents of the accumulator.

Example of a 3-byte instruction: LXI D,4FFFH
means Load a register pair (when an X appears as the second letter of an instruction mnemonic it means 'register pair') Immediate—the register pair being D and E [see the section on register pairs]—with the 16-bit hexadecimal data in the instruction. Assembled into machine code form this becomes

11 FF 4F

The first byte (11) is the opcode of the LXI instruction, the second byte (FF) is loaded into the 'low' register (register E) of the register pair DE, and the third byte (4F) is loaded into the 'high' register (register D) of the register pair DE.

Register addressing or register direct addressing

Register addressing is generally similar to direct addressing, but the data is stored in a register rather than in a memory location. The instruction specifies either a register in which an 8-bit data value is stored, or a register pair in which a 16-bit data value is stored.

Example involving a register: ADD B
means ADD the contents of register B to the accumulator, leaving the result in the accumulator. Assembled into machine code this becomes

80

Example involving a register pair: DAD D
means Double-length ADd the contents of the register pair DE to the contents of the register pair HL, leaving the result in HL. Assembled into machine code this becomes

19

Register indirect addressing

In this mode, a register pair contains a 16-bit address which is used as a memory address pointer; the data is stored in the address given by the pointer address.

Example: MOV B,M
means MOVe into register B the data in memory address M (whose address is specified by the 16-bit value in the register pair HL). For example, if the register pair HL contains the 16-bit hexadecimal value 4ABC, and location 4ABC contains the hexadecimal value 1F then, after the instruction is executed, register B contains 1F. Assembled into machine code this becomes

46

Implied or inherent addressing

The instruction itself implies the required address.

Example: STC
means SeT the Carry status flag; after the execution of this instruction, the C-flag contains logic '1'. Assembled into machine code form this becomes

37

Accumulator addressing

This is a form of implied addressing in which the accumulator is implied.

Example: CMA
means CoMplement the contents of the Accumulator; after the instruction is executed, the accumulator contains the 1's complement of its previous value. Assembled into machine code form this becomes

2F

Execution times

The execution time for each instruction is expressed in terms of the number of system clock cycles or states used for that instruction. In the case of conditional instructions including JUMP, CALL and RETURN instructions there are two possible cycle times, e.g., 7/10; which of the two values is used depends on the state of the condition flags.

Example:

Label	*Instruction mnemonic*	*Cycles*
	MVI D,DATA	;7
DELAY:	DCR D	;4
	JNZ DELAY	;EITHER 10 IF A JUMP ;OR 7 IF NO JUMP

The program calls for the DELAY loop to be executed (DATA − 1) times, after which an escape is made from the loop. The total *decimal* number of clock pulses needed before an escape is made from the program is given by

$$7 + ([4 + 10] \times [DATA - 1]) + [4 + 7]$$

where DATA in the above equation is a *decimal* value (the value of DATA in the program needs to be the *hexadecimal* equivalent of the decimal value in the equation). The first 7 in the equation corresponds to the MVI D,DATA instruction; the [4 + 10] corresponds to the DCR D and the JNZ DELAY instructions with a jump to DELAY; the [4 + 7] corresponds to the DCR D and the JNZ DELAY instructions when a jump does not occur.

Symbols and abbreviations

accumulator	A
Adr	16-bit address
D8	8-bit data
D16	16-bit data
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
Port	The 8-bit address of an I/O port
r, r1, r2	One of the registers A, B, C, D, E, H, L
rp	One of the register pairs: B represents B (high order) and C (low order) D represents D (high order) and E (low order) H represents H (high order) and L (low order)
M	The memory location whose address is contained in the register pair H and L
PSW	16-bit Processor Status Word comprising the 8-bit accumulator and the 8-bit flag register
SP	Represents the 16-bit Stack Pointer (sometimes treated as a register pair so far as programming is concerned)
SPH	High byte of SP content
SPL	Low byte of SP content
$((SP))_m$	Bit m of the byte on the current top of the stack
PC	Represents the 16-bit Program Counter
PCH	High byte of PC content
PCL	Low byte of PC content
rh	The high-order register of a rp
r1	The low-order register of a rp
( )	Means the contents of a memory location or a register
←	Means 'is transferred to'
↔	Means 'is exchanged with'
Λ	Logical AND
∀	Logical EXCLUSIVE–OR
V	Logical inclusive–OR
+	Addition
−	2's complement subtraction
*	Multiply
n	Restart number (0 through 7)

F — Flag register or status register comprising

CY CarrY flag
P Parity flag
AC Auxiliary Carry flag
Z Zero flag
S Sign flag

The format of the flag register is as follows

S	Z	X	AC	X	P	X	CY

where X = undefined

IM — Interrupt Mask register

The format of the data in the accumulator *before setting the interrupt mask* (SIM instruction) is

SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5

SOD (most significant bit) Serial Output Data
SOE Serial Output Enable
X undefined
R7.5 Reset RST 7.5
MSE Mask Set Enable
M7.5 RST 7.5 Mask
M6.5 RST 6.5 Mask
M5.5 RST 5.5 Mask

The format of the data in the accumulator *after reading the interrupt mask* (RIM instruction) is

SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

SID (most significant bit) Serial Input Data
I7.5 Pending Interrupt RST 7.5
I6.5 Pending Interrupt RST 6.5
I5.5 Pending Interrupt RST 5.5
IE Interrupt Enable flag
M7.5 RST 7.5 Mask
M6.5 RST 6.5 Mask
M5.5 RST 5.5 Mask

ACI: Add with Carry Immediate to the accumulator

Operation: (A) ← (A) + (byte 2) + (CY)

Description: The contents of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
ACI D8	7	2	CE

ADC: ADd register or memory contents with Carry to the accumulator contents

Operation: Register (A) ← (A) + (r) + (CY)
Memory (A) ← (A) + ((H) (L)) + (CY)

Description: Register. The contents of register r and the content of the CY flag are added to the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is contained in the register pair H and L and the content of the CY flag are added to the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
ADC A	4	1	8F
ADC B	4	1	88
ADC C	4	1	89
ADC D	4	1	8A
ADC E	4	1	8B
ADC H	4	1	8C
ADC L	4	1	8D
ADC M	7	1	8E

ADD: ADD register or memory contents to the accumulator

Operation: Register (A) ← (A) + (r)
Memory (A) ← (A) + ((H) (L))

Description: Register. The contents of register r are added to the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is contained in the register pair H and L are added to the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
ADD A	4	1	87
ADD B	4	1	80
ADD C	4	1	81
ADD D	4	1	82
ADD E	4	1	83
ADD H	4	1	84
ADD L	4	1	85
ADD M	7	1	86

ADI: ADd data Immediate to the accumulator contents

Operation: (A) ← (A) + (byte 2)

Description: The contents of the second byte of the instruction are added to the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
ADI D8	7	2	C6

ANA: ANd register or memory contents with Accumulator contents

Operation: Register (A) ← (A) Λ (r)
Memory (A) ← (A) Λ ((H) (L))

Description: Register. The contents of register r are logically ANDed with the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is contained in the register pair H and L are logically ANDed with the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags. The CY flag is cleared and AC is set.

Assembly language	Clock cycles	No. of bytes	Opcode
ANA A	4	1	A7
ANA B	4	1	A0
ANA C	4	1	A1
ANA D	4	1	A2
ANA E	4	1	A3
ANA H	4	1	A4
ANA L	4	1	A5
ANA M	7	1	A6

ANI: ANd data Immediate with the accumulator contents

Operation: (A) ← (A) Λ (byte 2)

Description: The contents of the second byte of the instruction are logically ANDed with the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags. The CY flag is cleared and AC is set.

Assembly language	Clock cycles	No. of bytes	Opcode
ANI D8	7	2	E6

CALL: CALL the subroutine whose address is given by the operand

Operation: ((SP) − 1) ← (PCH)
((SP) − 2) ← (PCL)
(SP) ← (SP) − 2
(PCL) ← (byte 2)
(PCH) ← (byte 3)

Description: The SP is decremented and the contents of PCH are stored in address (SP)−1; the SP is decremented once more and the contents of PCL are stored in address (SP)−2; on completion of the above, the SP is decremented by 2. Byte 2 of the instruction is moved to the low byte of the PC, and byte 3 is moved into the high byte of the PC.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
CALL Adr	18	3	CD

Ccondition: Call the subroutine whose address is given by the operand but only if a condition is satisfied

Operation: As for CALL if the specified condition is true.

Description: If the specified condition is true, the actions outlined in the CALL instructions are performed, otherwise control continues sequentially.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
CZ	9/18	3	CC
CNZ	9/18	3	C4
CC	9/18	3	DC
CNC	9/18	3	D4
CPE	9/18	3	EC
CPO	9/18	3	E4
CM	9/18	3	FC
CP	9/18	3	F4

CMA: CoMplement the Accumulator contents

Operation: (A) ← $\overline{(A)}$

Description: Each bit in the accumulator is logically complemented ('0' bits become 1's and '1' bits become 0's) the result remaining in the accumulator.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
CMA	4	1	2F

CMC: CoMplement the content of the Carry flag

Operation: (CY) ← $\overline{(CY)}$

Description: The bit in the CY flag is logically complemented.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
CMC	4	1	3F

CMP: CoMPare register or memory contents with the accumulator contents

Operation: Register (A) − (r)
Memory (A) − ((H) (L))

Description: Register. The contents of register r are subtracted from the contents of the accumulator, the result being discarded. The condition flags are set as a result of the subtraction.

Memory. The contents of the memory location whose address is contained in the H and L register pair are subtracted from the contents of the accumulator, the result being discarded. The condition flags are set as a result of the subtraction.

Flags which may be affected: All flags, in particular

CMP r the Z flag is set to '1' if (A) = (r)
the CY flag is set to '1' if (A) < (r)

CMP M the Z flag is set to '1' if (A) = ((H) (L))
the CY flag is set to '1' if (A) < ((H) (L))

Assembly language	Clock cycles	No. of bytes	Opcode
CMP A	4	1	BF
CMP B	4	1	B8
CMP C	4	1	B9
CMP D	4	1	BA
CMP E	4	1	BB
CMP H	4	1	BC
CMP L	4	1	BD
CMP M	7	1	BE

CPI: ComPare the accumulator contents Immediate with the data in the second byte of the instruction

Operation: (A) − (byte 2)

Description: The data in the second byte of the instruction is subtracted from the contents of the accumulator, the result being discarded. The condition flags are set as a result of the subtraction.

Flags which may be affected: All flags, in particular
the Z flag is set to '1' if (A) = (byte 2)
the CY flag is set to '1' if (A) < (byte 2)

Assembly language	Clock cycles	No. of bytes	Opcode
CPI D8	7	2	FE

DAA: Decimal Adjust the Accumulator contents

Description: The 8-bit value in the accumulator is adjusted to form two 4-bit Binary-Coded-Decimal (BCD) digits as follows.

1 If the value of the least significant four bits (b_3 to b_1) of the accumulator is greater than 9 OR if the AC flag is set (indicating a carry from b_3 to b_4), 6 is added to the least significant four bits of the accumulator.
2 If the value of the most significant four bits is greater than 9 OR if the CY flag is set, 6 is added to the most significant four bits of the accumulator.

Note: The DAA instruction is placed immediately after an add or a subtract instruction in the program.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
DAA	4	1	27

DAD: Double-length ADd the contents of a register pair to the register pair H and L

Operation: (H) (L) ← (H) (L) + (rh) (rl)

Description: The 16-bit contents of the register pair rp are added to the 16-bit contents of the register pair H and L. The 8-bit contents of rl are added to the 8-bit contents of register L, and the 8-bit contents of rh are added (with any carry from the less significant addition) to the 8-bit contents of register H, the result remaining in the register pair H and L.

Flags which may be affected: CY. If there is a carry-out from the 16-bit addition the CY flag is set to '1', otherwise it is cleared.

Assembly language	Clock cycles	No. of bytes	Opcode
DAD B	10	1	09
DAD D	10	1	19
DAD H	10	1	29
DAD SP	10	1	39

DCR: DeCRement register or memory contents

Operation: Register (r) ← (r) − 1
Memory ((H) (L)) ← ((H) (L)) − 1

Description: Register. The content of register r is decremented by unity.

Memory. The content of the memory location whose address is contained in the register pair H and L is decremented by unity.

Flags which may be affected: All except CY.

Assembly language	Clock cycles	No. of bytes	Opcode
DCR A	4	1	3D
DCR B	4	1	05
DCR C	4	1	0D
DCR D	4	1	15
DCR E	4	1	1D
DCR H	4	1	25
DCR L	4	1	2D
DCR M	10	1	35

DCX: DeCrement the contents of a register pair

Operation: (rh) (rl) ← (rh) (rl) − 1

Description: The 16-bit contents of the register pair rp are decremented by unity.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
DCX B	6	1	0B
DCX D	6	1	1B
DCX H	6	1	2B
DCX SP	6	1	3B

DI: Disable maskable Interrupts

Description: The maskable interrupt system is disabled immediately following the execution of this instruction.

Note: The maskable interrupt system is automatically disabled whenever an interrupt is acknowledged by the CPU.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
DI	4	1	F3

EI: Enable maskable Interrupts

Description: The maskable interrupt system is enabled but not until the next instruction in the program has been executed.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
EI	4	1	FB

HLT: HaLT

Description: The processor is stopped.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
HLT	5	1	76

IN: INput data to the accumulator from a port

Operation: (A) ← (Port)

Description: The data from the port whose address is specified by the second byte of the instruction is loaded into the accumulator.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
IN Port	10	2	DB

INR: INcRement register or memory

Operation: Register (r) ← (r) + 1
Memory ((H) (L)) ← ((H) (L)) + 1

Description: Register. The content of register r is incremented by unity.

Memory. The content of the memory location whose address is contained in the register pair H and L is incremented by unity.

Flags which may be affected: S, Z, AC, P.

Assembly language	Clock cycles	No. of bytes	Opcode
INR A	4	1	3C
INR B	4	1	04
INR C	4	1	0C
INR D	4	1	14
INR E	4	1	1C
INR H	4	1	24
INR L	4	1	2C
INR M	10	1	34

INX: INcrement a register pair

Operation: (rh) (rl) ← (rh) (rl) + 1

Description: The 16-bit content of the register pair rp is incremented by unity.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
INX B	6	1	03
INX D	6	1	13
INX H	6	1	23
INX SP	6	1	33

Jcondition: if a condition is satisfied, Jump to the address specified by the operand

Operation: (PCL) ← (byte 2)
(PCH) ← (byte 3)

Description: If the condition is true, program control is transferred to the address whose low byte is specified by byte 2 of the instruction and whose high byte is specified by byte 3.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
JZ Adr	7/10	3	CA
JNZ Adr	7/10	3	C2
JC Adr	7/10	3	DA
JNC Adr	7/10	3	D2
JPE Adr	7/10	3	EA
JPO Adr	7/10	3	E2
JM Adr	7/10	3	FA
JP Adr	7/10	3	F2

JMP: JuMP to the address given by the operand

Operation: (PCL) ← (byte 2)
(PCH) ← (byte 3)

Description: Program control is unconditionally transferred to the address whose low byte is specified by byte 2 of the instruction and whose high byte is given by byte 3.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
JMP Adr	10	3	C3

LDA: LoaD the Accumulator from a specified address

Operation: (A) ← ((byte 3) (byte 2))

Description: The contents of the memory location having the address specified as follows are loaded into the accumulator.
Low byte of address is in byte 2 of the instruction.
High byte of address is in byte 3 of the instruction.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
LDA Adr	13	3	3A

LDAX: LoaD the Accumulator from a memory location specified by the contents of a register pair

Operation: (A) ← ((rp))

Description: The contents of the memory location whose 16-bit address is stored in register pair rp, are loaded into the accumulator.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
LDAX B	7	1	0A
LDAX D	7	1	1A

LHLD: Load the register pair H and L Direct

Operation: (L) ← ((byte 3) (byte 2))
(H) ← ((byte 3) (byte 2) + 1)

Description: The contents of the memory location whose address is specified in byte 2 (low byte of address) and byte 3 (high byte of address) are loaded into register L. The contents of the memory location at the succeeding address are loaded into register H.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
LHLD	16	3	2A

LXI: Load into a register pair Immediate a 16-bit value

Operation: (rl) ← (byte 2)
(rh) ← (byte 3)

Description: Byte 2 of the instruction is loaded into the low-order register (rl) of a register pair, and byte 3 is loaded into the high-order register (rh).

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
LXI B,data 16	10	3	01
LXI D,data 16	10	3	11
LXI H,data 16	10	3	21
LXI SP,data 16	10	3	31

MOV: MOVe data between registers and/or memory

Operation: MOV register (r1) ← (r2)
MOV from memory (r) ← ((H) (L))
MOV to memory ((H) (L)) ← (r)

Description: MOV register. The contents of register r2 are moved to register r1.

MOV from memory. The contents of the memory location whose 16-bit address is stored in the register pair H and L are moved to register r.

MOV to memory. The contents of register r are moved to the memory location specified by the 16-bit address contained in the register pair H and L.

Flags which may be affected: None.

Note:
Register-to-register MOV instructions use 4 clock cycles.

Register-to-memory and memory-to-register instructions use 7 clock cycles.

Each MOV instruction is one byte in length.

Assembly language	Opcode	Assembly language	Opcode
MOV A,A	7F	MOV B,A	47
MOV A,B	78	MOV B,B	40
MOV A,C	79	MOV B,C	41
MOV A,D	7A	MOV B,D	42
MOV A,E	7B	MOV B,E	43
MOV A,H	7C	MOV B,H	44
MOV A,L	7D	MOV B,L	45
MOV A,M	7E	MOV B,M	46
MOV C,A	4F	MOV D,A	57
MOV C,B	48	MOV D,B	50
MOV C,C	49	MOV D,C	51
MOV C,D	4A	MOV D,D	52
MOV C,E	4B	MOV D,E	53
MOV C,H	4C	MOV D,H	54
MOV C,L	4D	MOV D,L	55
MOV C,M	4E	MOV D,M	56

Assembly language	Opcode
MOV E,A	5F
MOV E,B	58
MOV E,C	59
MOV E,D	5A
MOV E,E	5B
MOV E,H	5C
MOV E,L	5D
MOV E,M	5E
MOV L,A	6F
MOV L,B	68
MOV L,C	69
MOV L,D	6A
MOV L,E	6B
MOV L,H	6C
MOV L,L	6D
MOV L,M	6E

Assembly language	Opcode
MOV H,A	67
MOV H,B	60
MOV H,C	61
MOV H,D	62
MOV H,E	63
MOV H,H	64
MOV H,L	65
MOV H,M	66
MOV M,A	77
MOV M,B	70
MOV M,C	71
MOV M,D	72
MOV M,E	73
MOV M,H	74
MOV M,L	75

MVI: MoVe data Immediate into a register or memory

Operation: Register (r) ← (byte 2)
Memory ((H) (L)) ← (byte 2)

Description: Register. The contents of byte 2 of the instruction are moved immediate into register r.

Memory. The contents of byte 2 of the instruction are moved immediate into the memory location given by the 16-bit value in the register pair H and L.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
MVI A,D8	7	2	3E
MVI B,D8	7	2	06
MVI C,D8	7	2	0E
MVI D,D8	7	2	16
MVI E,D8	7	2	1E
MVI H,D8	7	2	26
MVI L,D8	7	2	2E
MVI M,D8	10	2	36

NOP: NO oPeration

Description: No operation is performed by the CPU.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
NOP	4	1	00

ORA: OR the contents of a register or a memory with the Accumulator contents

Operation: Register (A) ← (A) V (r)
Memory (A) ← (A) V ((H) (L))

Description: Register. The contents of register r are inclusive–ORed with the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of a memory location whose address is given by the 16-bit value in the H and L register pair are inclusive–ORed with the content of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags. The AC and CY flags are cleared.

Assembly language	Clock cycles	No. of bytes	Opcode
ORA A	4	1	B7
ORA B	4	1	B0
ORA C	4	1	B1
ORA D	4	1	B2
ORA E	4	1	B3
ORA H	4	1	B4
ORA L	4	1	B5
ORA M	7	1	B6

ORI: OR data Immediate with the accumulator content

Operation: (A) ← (A) V (byte 2)

Description: The contents of the second byte of the instruction are inclusive–ORed with the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags. The AC and CY flags are cleared.

Assembly language	Clock cycles	No. of bytes	Opcode
ORI D8	7	2	F6

OUT: OUTput data from the accumulator to a port

Operation: (Port) ← (A)

Description: The content of the accumulator is output to the port specified in the second byte of the instruction.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
OUT port	10	2	D3

PCHL: replace the contents of PC by the contents of HL (jump to the address specified by the contents of the register pair H and L)

Operation: (PCH) ← (H)
(PCL) ← (L)

Description: The contents of register H are moved to the high-order eight bits of the 16-bit PC, and the contents of register L are moved to the low-order 8-bits of the PC.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
PCHL	6	1	E9

POP: POP two bytes of data from the current top of the stack either to a specified register pair or to the processor status word (PSW)

Operation: Register pair (rl) ← ((SP))
(rh) ← ((SP) + 1)
(SP) ← (SP) + 2
Processor status word (PSW) (CY) ← $((SP))_0$
(P) ← $((SP))_2$
(AC) ← $((SP))_4$
(Z) ← $((SP))_6$
(S) ← $((SP))_7$
(A) ← ((SP) + 1)
(SP) ← (SP) + 2

Description: Register pair. The contents of the memory location whose address is specified by the contents of the SP are moved to the low-order register of the register pair rp. The contents of the next higher address are moved to the high-order register of the register pair rp. The SP is incremented by two.

Processor status word. The contents of the memory location whose address is specified by the contents of the SP are used to restore data to the flag register. The contents of the next higher address are moved to the accumulator. The SP is incremented by two.

Flags which may be affected: Register pair—none.
Processor status word—all.

Assembly language	Clock cycles	No. of bytes	Opcode
POP B	10	1	C1
POP D	10	1	D1
POP H	10	1	E1
POP PSW	10	1	F1

PUSH: PUSH the 2-byte contents of a register pair or the PSW onto the current top of the stack

Operation: Register pair $((SP) - 1) \leftarrow (rh)$
$((SP) - 2) \leftarrow (rl)$
$(SP) \leftarrow (SP) - 2$

Processor status word (PSW) $((SP) - 1) \leftarrow (A)$
$((SP) - 2)_0 \leftarrow (CY)$
$((SP) - 2)_2 \leftarrow (P)$
$((SP) - 2)_4 \leftarrow (AC)$
$((SP) - 2)_6 \leftarrow (Z)$
$((SP) - 2)_7 \leftarrow (S)$
$(SP) \leftarrow (SP) - 2$

Description: Register pair. The contents of the high-order register of the register pair rp are moved to the memory location whose address is one less than the contents of the SP. The contents of the low-order register of the register pair rp are moved to the memory location whose address is two less than the contents of the SP. The SP is decremented by two.

Processor status word. The contents of the accumulator are moved to the memory location whose address is one less than the contents of the SP. The contents of the flag register are moved to the memory location whose address is two less than the contents of the SP. The SP is decremented by two.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
PUSH B	12	1	C5
PUSH D	12	1	D5
PUSH H	12	1	E5
PUSH PSW	12	1	F5

RAL: Rotate Accumulator contents Left through the carry flag

Operation: $(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$; $(A_0) \leftarrow (CY)$

Description: The contents of the accumulator are rotated left one position, the high-order bit is moved into the CY flag, and the low-order bit is set equal to the CY flag.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
RAL	4	1	17

RAR: Rotate Accumulator contents Right through the carry flag

Operation: $(A_n) \leftarrow (A_{n+1})$; $(CY) \leftarrow (A_0)$; $(A_7) \leftarrow (CY)$

Description: The contents of the accumulator are rotated right one bit, the low-order bit is moved into the CY flag, and the high-order bit is set equal to the CY flag.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
RAR	4	1	1F

Rcondition: Return to the calling program if a condition is satisfied

Operation: If the condition is satisfied then
(PCL) ← ((SP))
(PCH) ← ((SP) + 1)
(SP) ← (SP) + 2

Description: If the specified condition is true, the contents of the memory location whose address is specified by the 16-bit value in the SP are moved to the low-order byte of the PC. The contents of the memory location which is one higher than the SP are moved to the high-order byte of the PC. The SP is incremented by two. If the specified condition is not true, control continues sequentially.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
RZ	6/12	1	C8
RNZ	6/12	1	C0
RC	6/12	1	D8
RNC	6/12	1	D0
RPE	6/12	1	E8
RPO	6/12	1	E0
RM	6/12	1	F8
RP	6/12	1	F0

RET: RETurn from subroutine

Operation: (PCL) ← ((SP))
(PCH) ← ((SP) + 1)
(SP) ← (SP) + 2

Description: The actions listed under 'description' for the Rcondition instructions are performed unconditionally.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
RET	10	1	C9

RIM: Read Interrupt Mask register

Operation: (A) ← (IM)

Description: The contents of the interrupt mask register (see **Symbols and abbreviations**) are moved into the accumulator.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
RIM	4	1	20

RLC: Rotate Left the accumulator contents with a branch to the Carry flag

Operation: $(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$; $(CY) \leftarrow (A_7)$

Description: The contents of the accumulator are 'rotated' left one position, the low-order bit and the CY flag are set to the value shifted out of the high-order bit.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
RLC	4	1	07

RRC: Rotate Right the accumulator contents with a branch to the Carry flag

Operation: $(A_{n-1}) \leftarrow (A_n)$; $(A_7) \leftarrow (A_0)$; $(CY) \leftarrow (A_0)$

Description: The contents of the accumulator are 'rotated' right one position, the high-order bit and the CY flag are set to the value shifted out of the low-order bit.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
RRC	4	1	0F

RST: ReSTart the program at a specified (interrupt) address

Operation: ((SP) − 1) ← (PCH)
((SP) − 2) ← (PCL)
(SP) ← (SP) − 2
(PCH) ← 00H
(PCL) ← $[8 * n]_{16}$

Description: The high byte of the PC is moved into the memory location whose address is one less than the contents of the SP register. The low byte of the PC is moved into the memory location whose address is two less than the contents of the SP. The SP is decremented twice. Zero (00H) is moved into the high byte of the PC, and a hexadecimal value equal to eight times the restart number n is moved into the low byte of the PC. Program control is transferred to the address given by the value in the PC.

Note: The following table includes the address to which the CPU is vectored in the event of an interrupt occurring. Restart 7.5, 6.5, 5.5 and 4.5 (the latter being the TRAP signal) are hardware interrupts.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode	Vector address (hex)
RST 0	12	1	C7	0000
RST 1	12	1	CF	0008
RST 2	12	1	D7	0010
RST 3	12	1	DF	0018
RST 4	12	1	E7	0020
RST 5	12	1	EF	0028
RST 6	12	1	F7	0030
RST 7	12	1	FF	0038
RST 4.5 [TRAP]			hardware	0024
RST 5.5			hardware	002C
RST 6.5			hardware	0034
RST 7.5			hardware	003C

SBB: SuBtract register or memory contents from the accumulator contents with Borrow

Operation: Register (A) ← (A) − (r) − (CY)
Memory (A) ← (A) − ((H) (L)) − (CY)

Description: Register. The contents of register r and the content of the CY flag are both subtracted from the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is contained in the H and L register pair and the content of the CY flag are both subtracted from the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
SBB A	4	1	9F
SBB B	4	1	98
SBB C	4	1	99
SBB D	4	1	9A
SBB E	4	1	9B
SBB H	4	1	9C
SBB L	4	1	9D
SBB M	7	1	9E

SBI: SuBtract data Immediate from the accumulator with borrow

Operation: (A) ← (A) − (byte 2) − (CY)

Description: The contents of the second byte of the instruction and the content of the CY flag are both subtracted from the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
SBI data	7	2	DE

SHLD: Store the contents of the register pair H and L Direct

Operation: ((byte 3) (byte 2)) ← (L)
((byte 3) (byte 2) + 1) ← (H)

Description: The contents of register L are moved to the memory location whose address is given by byte 2 (low byte of the destination address) and byte 3 (high byte of the destination address) of the instruction. The contents of register H are stored in the next higher memory location.

Assembly language	Clock cycles	No. of bytes	Opcode
SHLD Adr	16	3	22

SIM: Set Interrupt Mask

Operation: (IM) ← (A)

Description: The contents of the accumulator are moved into the interrupt mask (*see also* **Symbols and abbreviations**).

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
SIM	4	1	30

SPHL: load the Stack Pointer from the H and L register pair

Operation: (SPL) ← (L)
(SPH) ← (H)

Description: The contents of register L are moved into the low byte of the SP, and the contents of register H are moved into the high byte of the SP.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
SPHL	6	1	F9

STA: STore Accumulator direct in a memory location

Operation: ((byte 3) (byte 2)) ← (A)

Description: The contents of the accumulator are moved into the memory location whose address is specified by byte 2 (low byte of the destination address) and byte 3 (high byte of the destination address) of the instruction.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
STA addr	13	3	32

STAX: STore the Accumulator contents in the memory location addressed by a register pair

Operation: ((rp)) ← (A)

Description: The contents of the accumulator are moved into the memory location whose address is given by the 16-bit value in the register pair rp.

Note: only the register pair B (registers B and C) and D (registers D and E) can be specified.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
STAX B	7	1	02
STAX D	7	1	12

STC: SeT the Carry flag to '1'

Operation: (CY) ← 1

Description: The CY flag is set to '1'.

Flags which may be affected: CY.

Assembly language	Clock cycles	No. of bytes	Opcode
STC	4	1	37

SUB: SUBtract register or memory contents from the accumulator

Operation: Register (A) ← (A) − (r)
Memory (A) ← (A) − ((H) (L))

Description: Register. The contents of register r are subtracted from the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is contained in the H and L register pair are subtracted from the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
SUB A	4	1	97
SUB B	4	1	90
SUB C	4	1	91
SUB D	4	1	92
SUB E	4	1	93
SUB H	4	1	94
SUB L	4	1	95
SUB M	7	1	96

SUI: SUbtract data Immediate from the accumulator contents

Operation: (A) ← (A) − (byte 2)

Description: The data in the second byte of the instruction is subtracted from the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags.

Assembly language	Clock cycles	No. of bytes	Opcode
SUI D8	7	2	D6

XCHG: eXCHange the contents of the register pair D and E with the contents of register pair H and L

Operation: (H) ⟷ (D); (L) ⟷ (E)

Description: The contents of registers H and D are exchanged as are the contents of registers L and E.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
XCHG	4	1	EB

XRA: eXclusive–oR the contents of a register or of a memory with the Accumulator contents

Operation: Register (A) ← (A) ∀ (r)
Memory (A) ← (A) ∀ ((H) (L))

Description: Register. The contents of register r are EXCLUSIVE–ORed with the contents of the accumulator, the result remaining in the accumulator.

Memory. The contents of the memory location whose address is given by the 16-bit value in the register pair H and L are EXCLUSIVE–ORed with the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: All flags. The AC and CY flags are cleared.

Assembly language	Clock cycles	No. of bytes	Opcode
XRA A	4	1	AF
XRA B	4	1	A8
XRA C	4	1	A9
XRA D	4	1	AA
XRA E	4	1	AB
XRA H	4	1	AC
XRA L	4	1	AD
XRA M	7	1	AE

XRI: eXclusive–oR data Immediate with the accumulator content

Operation: (A) ← (A) ∀ (byte 2)

Description: The data in the second byte of the instruction is EXCLUSIVE–ORed with the contents of the accumulator, the result remaining in the accumulator.

Flags which may be affected: ALL flags. The AC and CY flags are cleared.

Assembly language	Clock cycles	No. of bytes	Opcode
XRI D8	7	2	EE

XTHL: eXchange Top of the stack with the contents of the H and L registers

Operation: (L) ⟷ ((SP)); (H) ⟷ ((SP)+1)

Description: The content of the L register is exchanged with the content of the memory location whose address resides in the SP. The content of register L is exchanged with the content of the memory location whose address is one higher than the address in the SP.

Flags which may be affected: None.

Assembly language	Clock cycles	No. of bytes	Opcode
XTHL	16	1	E3

Operation codes in numerical sequence

The following expressions are used in this section

Adr	16-bit address
D8	An 8-bit data value
D16	A 16-bit data value
Port	An 8-bit port address

Opcode	Assembly language	Opcode	Assembly language
00	NOP	1F	RAR
01	LXI B,D16	20	RIM
02	STAX B	21	LXI H,D16
03	INX B	22	SHLD Adr
04	INR B	23	INX H
05	DCR B	24	INR H
06	MVI B,D8	25	DCR H
07	RLC	26	MVI H,D8
08		27	DAA
09	DAD B	28	
0A	LDAX B	29	DAD H
0B	DCX B	2A	LHLD Adr
0C	INR C	2B	DCX H
0D	DCR C	2C	INR L
0E	MVI C,D8	2D	DCR L
0F	RRC	2E	MVI L,D8
10		2F	CMA
11	LXI D,D16	30	SIM
12	STAX D	31	LXI SP,D16
13	INX D	32	STA Adr
14	INR D	33	INX SP
15	DCR D	34	INR M
16	MVI D,D8	35	DCR M
17	RAL	36	MVI M,D8
18		37	STC
19	DAD D	38	
1A	LDAX D	39	DAD SP
1B	DCX D	3A	LDA Adr
1C	INR E	3B	DCX SP
1D	DCR E	3C	INR A
1E	MVI E,D8	3D	DCR A

Opcode	Assembly language	Opcode	Assembly language
3E	MVI A,D8	64	MOV H,H
3F	CMC	65	MOV H,L
40	MOV B,B	66	MOV H,M
41	MOV B,C	67	MOV H,A
42	MOV B,D	68	MOV L,B
43	MOV B,E	69	MOV L,C
44	MOV B,H	6A	MOV L,D
45	MOV B,L	6B	MOV L,E
46	MOV B,M	6C	MOV L,H
47	MOV B,A	6D	MOV L,L
48	MOV C,B	6E	MOV L,M
49	MOV C,C	6F	MOV L,A
4A	MOV C,D	70	MOV M,B
4B	MOV C,E	71	MOV M,C
4C	MOV C,H	72	MOV M,D
4D	MOV C,L	73	MOV M,E
4E	MOV C,M	74	MOV M,H
4F	MOV C,A	75	MOV M,L
50	MOV D,B	76	HLT
51	MOV D,C	77	MOV M,A
52	MOV D,D	78	MOV A,B
53	MOV D,E	79	MOV A,C
54	MOV D,H	7A	MOV A,D
55	MOV D,L	7B	MOV A,E
56	MOV D,M	7C	MOV A,H
57	MOV D,A	7D	MOV A,L
58	MOV E,B	7E	MOV A,M
59	MOV E,C	7F	MOV A,A
5A	MOV E,D	80	ADD B
5B	MOV E,E	81	ADD C
5C	MOV E,H	82	ADD D
5D	MOV E,L	83	ADD E
5E	MOV E,M	84	ADD H
5F	MOV E,A	85	ADD L
60	MOV H,B	86	ADD M
61	MOV H,C	87	ADD A
62	MOV H,D	88	ADC B
63	MOV H,E	89	ADC C

Opcode	Assembly language	Opcode	Assembly language
8A	ADC D	B0	ORA B
8B	ADC E	B1	ORA C
8C	ADC H	B2	ORA D
8D	ADC L	B3	ORA E
8E	ADC M	B4	ORA H
8F	ADC A	B5	ORA L
90	SUB B	B6	ORA M
91	SUB C	B7	ORA A
92	SUB D	B8	CMP B
93	SUB E	B9	CMP C
94	SUB H	BA	CMP D
95	SUB L	BB	CMP E
96	SUB M	BC	CMP H
97	SUB A	BD	CMP L
98	SBB B	BE	CMP M
99	SBB C	BF	CMP A
9A	SBB D	C0	RNZ
9B	SBB E	C1	POP B
9C	SBB H	C2	JNZ Adr
9D	SBB L	C3	JMP Adr
9E	SBB M	C4	CNZ Adr
9F	SBB A	C5	PUSH B
A0	ANA B	C6	ADI D8
A1	ANA C	C7	RST 0
A2	ANA D	C8	RZ
A3	ANA E	C9	RET
A4	ANA H	CA	JZ Adr
A5	ANA L	CB	
A6	ANA M	CC	CZ Adr
A7	ANA A	CD	CALL Adr
A8	XRA B	CE	ACI D8
A9	XRA C	CF	RST 1
AA	XRA D	D0	RNC
AB	XRA E	D1	POP D
AC	XRA H	D2	JNC Adr
AD	XRA L	D3	OUT Port
AE	XRA M	D4	CNC Adr
AF	XRA A	D5	PUSH D

Opcode	Assembly language	Opcode	Assembly language
D6	SUI D8	EB	XCHG
D7	RST 2	EC	CPE Adr
D8	RC	ED	
D9		EE	XRI D8
DA	JC Adr	EF	RST 5
DB	IN Port	F0	RP
DC	CC Adr	F1	POP PSW
DD		F2	JP Adr
DE	SBI D8	F3	DI
DF	RST 3	F4	CP Adr
E0	RPO	F5	PUSH PSW
E1	POP H	F6	ORI D8
E2	JPO Adr	F7	RST 6
E3	XTHL	F8	RM
E4	CPO Adr	F9	SPHL
E5	PUSH H	FA	JM Adr
E6	ANI D8	FB	EI
E7	RST 4	FC	CM Adr
E8	RPE	FD	
E9	PCHL	FE	CPI D8
EA	JPE Adr	FF	RST 7

8085 instruction set classification

The 8085 instruction set can be classified into five different types of instruction as follows:

Data transfer group—move data between registers or between memory and registers.
Arithmetic group—add, subtract, increment or decrement data in registers or in memory.
Logical group—AND, OR, EXCLUSIVE–OR, compare, rotate or complement data in registers or in memory.
Branch group—conditional and unconditional jump instructions and return instructions.
Stack, I/O and machine control group—stack control, I/O and internal flag control.
The groups of instructions are detailed below.

Data transfer group

MOV r1,r2	MOV r,M	MOV M,r	MVI r,D8
MVI M,D8	LXI rp,D16	LDA Adr	STA Adr
LHLD Adr	SHLD Adr	LDAX rp	STAX rp
XCHG			

Arithmetic group

ADD r*	ADD M*	ADI D8*	ADC r*
ADC M*	ACI D8*	SUB r*	SUB M*
SUI D8*	SBB r*	SBB M*	SBI D8*
INR r**	INR M**	DCR r**	DCR M**
INX rp	DCX rp	DAD rp***	DAA*

Logical group

ANA r*	ANA M*	ANI D8*	XRA r*
XRA M*	XRI D8*	ORA r*	ORA M*
ORI D8*	CMP r*	CMP M*	CPI D8*
RLC	RRC	RAL	RAR
CMA	CMC***	STC***	

Branch group

JMP Adr	Jcondition Adr	RET	Rcondition
RST n	PCHL	CALL Adr	Ccondition Adr

Stack, I/O and machine control group

PUSH rp	PUSH PSW	POP rp	POP PSW
XTHL	SPHL	IN Port	OUT Port
EI	DI	HLT	NOP
RIM	SIM		

Note: * means all flags may be affected
** means all flags except carry may be affected
*** only carry flag affected

Stack processes

The stack is an area of RAM where data can be temporarily stored by the CPU, and is organized in a **Last-In, First-Out** (LIFO) mode; the stack is used in association with subroutines and interrupts.

In the 8085, the stack can be established at any point in the usable RAM, and can have any 'length' (subject to the amount of RAM available); the address of the current top of the stack is stored in the 16-bit **stack pointer** register, SP. By using the SPHL instruction, it is possible for the user to access two independent stacks.

It is good programming practice to initialize the address of the top of the stack in the SP register at an early point in the program by means of a

LXI SP,D16

instruction. For example

LXI SP,4FFFH

initializes the address of the top of the stack to the location 4FFFH.

Data is pushed onto the stack two bytes at a time, each byte occupying successive memory locations below the top of the stack. The two bytes of data transferred to the stack are fetched either from a register pair or from the PSW. The SP is decremented *before* each byte of data is pushed onto the stack; this means that the initial address in the SP is not used for data storage (that is, it is not part of the stack memory 'area' so far as data storage is concerned).

PUSH operations

Figure 2 illustrates the mechanism of the stack operation. Suppose that the SP is initialized to 20B0H by means of an

LXI SP,20B0H

instruction and also that register B contains byte B of data, register C contains byte C, register D contains byte D, and register E contains byte E.

When the instruction PUSH B is encountered, the contents of register B

Address	Memory content
4FFF	XX
4FFE	byte B
4FFD	byte C
4FFC	byte D
4FFB	byte E
4FFA	XX

Fig. 2

(byte B) is PUSHed into location (4FFF − 1) = 4FFEH, and the contents of register C (byte C) is pushed into location (4FFF − 2) = 4FFDH. The SP is then decremented by two so that the SP contains the address (4FFF − 2) = 4FFDH *after the PUSH B instruction is completed*; the latter address is the new 'top' of the stack. The reader will note that the location at the initial top of the stack (address 4FFFH) contains irrelevant electronic 'garbage'.

When a PUSH D instruction is encountered, the contents of register D (byte D) are PUSHed into location (4FFD − 1) = 4FFCH, and the contents of register E (byte E) are pushed into location 4FFBH. The SP is decremented by two again, to give a new top of the stack address of 4FFBH.

POP operations

Data is retrieved from the stack two bytes at a time, each of the bytes being POPed into one register of a register pair (or into the PSW). POP instructions are used in the program at a later point in the program than PUSH instructions. Let us assume that the SP was initialized at 4FFFH early in the program, and that the instructions PUSH B and PUSH D have been executed (see *also* Figure 2); the SP contains 4FFBH at this point.

When a POP D instruction is encountered, the following events occur. Firstly, byte E in the address given by the SP (4FFBH) is POPed into the low-order register of the register pair (register E), and byte D in the next higher address (4FFCH) is POPed into the high-order register of the register pair (register D); the SP is incremented by two to give a new top of stack address of (4FFB + 2) = 4FFDH.

When a POP B instruction is encountered, byte C at the new top of stack address (4FFDH) is POPed into register C, and byte B in address 4FFEH is POPed into register B; the SP is incremented by two to give the top of stack address in the SP as (20AE + 2) = 4FFFH.

PUSH and POP sequence

Care must be taken when using PUSH and POP instructions to ensure that the register and stack contents are not corrupted. The programmer must ensure that the POP sequence is the mirror *image* of the PUSH sequence. For example, if the programmer

wishes to 'save' the status of every register in the CPU on the stack and, at some later time, restore the same data to the registers, a sequence of instructions similar to those given below must be executed.

```
PUSH B      ;SAVE B AND C
PUSH D      ;SAVE D AND E
PUSH H      ;SAVE H AND L
PUSH PSW    ;SAVE A AND FLAGS
....
....        }    PROGRAM CONTINUES
....
POP PSW     ;RESTORE FLAGS AND A
POP H       ;RESTORE L AND H
POP D       ;RESTORE E AND D
POP B       ;RESTORE C AND B
```

However, should it be necessary at any time to interchange the contents of one register pair with another (see *also* XTHL instruction), then the PUSH and POP sequence can be reversed. For example, the following instructions reverse the register contents:

```
PUSH B
PUSH D
POP B
POP D
```

In other words, on the last in first out principle, the bytes PUSHed onto the stack from register pairs D and E are the first to be POPed into the register pairs B and C. The former contents of registers B and C are then POPed into registers D and E, thus exchanging their original contents.

Subroutines

A subroutine is a 'self contained' subprogram which can be entered from any point in the main program. Program control can be transferred unconditionally to a subroutine by means of a 3-byte CALL Adr instruction, e.g., CALL 4020H. The latter instruction has the effect of transferring control to address 4020H, the return address to the calling program meanwhile being saved on the top of the stack. The subroutine may either be given an

absolute address such as 2040H (above), or can be given a *symbolic address* such as SUBR (in which case the actual address of SUBR will have been established by means of an equate directive such as SUBR EQU 4020H). The CPU interprets a CALL SUBR instruction in the same way as a CALL 4020H instruction. A typical program involving a subroutine is given below.

```
2000          LXI SP,4FFFH  ;INITIALIZE SP
....          ....          ;MAIN PROGRAM
2017          CALL 4020H    ;CALL SUBROUTINE
201A          STA 2000H     ;MAIN PROGRAM CONTINUES
....          ....
2200          HLT           ;HALT MAIN PROGRAM
....          ....
....          ....
4020  SUBR:   PUSH B       }
              PUSH D        }  SAVE CPU STATUS
              PUSH H        }
              PUSH PSW      }
              ....          }
              ....          }  MAIN BODY OF SUBROUTINE
              ....          }
              POP PSW       }
              POP H         }
              POP D         }  RESTORE CPU STATUS
              POP B         }
              RET           ;RETURN TO CALLING PROGRAM
```

The first instruction in the program initializes the SP to 4FFFH and the program continues until, at address 2017H, the CPU encounters a CALL 4020H instruction. This instruction causes the address 4020 to be moved to the PC, causing program control to be transferred to that address. At the same time, it results in a 'transparent' PUSH operation to be performed in which the 16-bit address of the next instruction in the calling program (the main program in this case) is PUSHed onto the two locations at the top of the stack.

This is illustrated in Figure 3, in which the high byte of the return address is pushed into the location which is one below the value stored in the SP, i.e., into address (4FFF − 1) = 4FFEH, and the low byte of the return address is PUSHed into location (4FFF − 2) = 4FFDH. In this way, the return address to the calling program is saved on the stack. The transparent PUSH operation is performed in much the same way as a conventional PUSH operation, and the SP is decremented by two on completion of the instructions so that the new top of the stack is (4FFF − 2) = 4FFDH.

Address	Memory contents
4FFF	XX
4FFE	20
4FFD	1A
4FFC	(REG B)
4FFB	(REG C)
....	....
4FF6	(REG A)
4FF5	(FLAGS)
4FF4	XX

Fig. 3

A subroutine can also be CALLed conditionally (see Ccondition instructions), which results in control being transferred to the subroutine if a particular condition is satisfied.

A subroutine may contain up to four sections, illustrated in the example subroutine program earlier which commenced at address 4020H, and are described below.

1 *Save status* section, in which the contents of the CPU registers are saved on the stack.
2 *Main body* of the subroutine, in which the principal function of the subroutine is performed, e.g., multiplication.
3 *Restore status* section, in which the original contents of the registers are restored.
4 *Return* to the calling program.

Sections 1 and 3 are optional, and need not be included in every subroutine. Figure 3 illustrates the contents of the stack addresses from 4FFFH to 4FF4H after the 'save CPU status' section is complete. At this stage the contents of the flag register are stored in location 4FF5H, which is the top of the stack at the time the main body of the subroutine is entered. If the stack is used during the main body of the subroutine, addresses 4FF4H and lower are used; it is the responsibility of the programmer to ensure that, on completion of the main part of the subroutine, the SP contains address 4FF5H. If the stack is not used during the main part, the SP contains 4FF5H until the restore status section is entered.

During the 'restore CPU status' section, the contents of locations 4FF5H to 4FFCH are restored to the correct registers by means of a series of POP instructions.

Prior to the RETurn instruction being executed, the SP contains 4FFDH, which is the location storing the low byte of the return address to the main program. The RETurn instruction results in the following operations.

1 The two bytes at the top of the stack are transferred to the PC, i.e., 201AH is moved into the PC.
2 The SP is incremented by two, i.e., the SP stores 4FFFH.

Control is therefore transferred to the instruction in location 201AH, after which the program continues sequentially (unless the subroutine is CALLed again).

A return can be made conditionally from the subroutine by means of one of the Rcondition instructions described in the *assembly language* section.

Nested subroutines

'Nesting' is a process by which one subroutine can call another subroutine, which can call a third, etc. An example of a program having a nesting depth of two is given below.

```
2000    MAIN:   LXI SP,4FFFH    ;INITIALIZE SP
....            ....            ;MAIN PROGRAM
2021            CALL SUB 1      ;CALL SUBROUTINE 1
2024            MVI A,00H       ;MAIN PROGRAM CONTINUES
....            ....
206F            HLT             ;END OF MAIN PROGRAM
....            ....
2080    SUB 1:  PUSH B          ;SUBROUTINE 1 COMMENCES
....            ....
208B            CALL SUB 2      ;CALL SUBROUTINE 2
208E            MVI A,01H       ;SUBROUTINE 1 CONTINUES
....            ....
2099            RET             ;RETURN TO MAIN PROGRAM
....            ....
20B0    SUB 2:  PUSH B          ;SUBROUTINE 2 COMMENCES
....            ....
20C0            RET             ;RETURN TO SUBROUTINE 1
```

Interrupts

When an interrupt signal (which is logic '1' in the case of the 8085) is applied to an interrupt pin on the CPU (or, alternatively, an RST n instruction is executed), the normal sequence of operations in the main program (or in a subroutine if one is being executed) is interrupted; program control is transferred or 'vectored' to an address in memory which directs the CPU to a location where an 'interrupt' routine is to be found. An interrupt routine is generally similar to a subroutine, but has additional features described below. On completion of the interrupt routine, program control is transferred to the main program in much the same way as for a subroutine.

The 8085 has four *hardware* interrupt pins (RST 4.5 [TRAP], RST 5.5, RST 6.5, and RST 7.5) on the CPU chip and eight software/hardware interrupts (RST 0 through RST 7)—the hardware versions of the latter utilizing another interrupt pin—INTR—on the CPU chip. When not in use, each hardware pin is held at logic '0'.

An interrupt is one of two types, namely

1 A maskable interrupt.
2 A nonmaskable interrupt.

A **maskable interrupt** may be masked out by means of an instruction, that is, the CPU can ignore the interrupt request if the programmer wishes it. A **nonmaskable interrupt** cannot be masked out by an instruction. The 8085 has one nonmaskable interrupt (which is the TRAP or RST 4.5 interrupt) and several maskable interrupts.

When a peripheral applies a logic '1' to any one of the RST 4.5, RST 5.5, RST 6.5 or RST 7.5 interrupt lines (thereby indicating that it needs to be serviced), the CPU completes the current instruction and then executes an internal RST operation which causes

1 the contents of the PC to be saved on the stack and
2 a branch to be made to the appropriate ReSTart address in memory.

Each ReSTart address in memory has only a few bytes of memory space allocated to it (see RST instructions); it is generally the case that each of these vector addresses contains a 3-byte unconditional JUMP instruction. The latter instruction transfers control to another address which has enough memory space associated with it to accommodate the interrupt routine.

The interrupts should be enabled at an early point in the program, thereby allowing the main program to be interrupted by a peripheral at any time when an interrupt request is made. Since the 8085 has a number of maskable interrupting sources (RST 5.5, RST 6.5 and RST 7.5), it is necessary to **unmask** interrupts which are to be allowed and to **mask out** those which are not. The latter process is performed by **setting the interrupt mask** (see the SIM instruction and also the interrupt mask in **Symbols and abbreviations**) as shown below. Following this, the entire maskable interrupt system is enabled as follows.

```
LXI SP,4FFFH      ;INITIALIZE SP
....              ;REMAINDER OF INITIALIZATION PART OF
....              ;PROGRAM
MVI A,0DH         ;UNMASK RST 6.5, AND MASK OUT
SIM               ;RST 5.5 AND RST 7.5
EI                ;ENABLE INTERRUPTS
....              ;MAIN PROGRAM CONTINUES
```

The first instruction in the above program initializes the SP, after which the interrupts are either allowed (unmasked) or disallowed (masked out) by moving one of the following values into the appropriate bit of the interrupt mask

0 unmasks (allows) an interrupt to occur
1 masks out (disallows) an interrupt.

Interrupt RST 6.5 is unmasked and interrupts RST 5.5 and 7.5 are masked out by transferring the value 00001101_2 (or 0DH) into the accumulator by means of the MVI A,0DH instruction, and then transferring it into the interrupt mask by using the SIM instruction. Referring to the interrupt mask register (see **Symbols and abbreviations**), the reader will note that this places a '0' in the M6.5 position in the register, and a '1' in the M5.5, M7.5 and MSE positions; this results in RST 6.5 being unmasked and the remaining maskable interrupts being masked out. It is also necessary for the MSE bit to be logic '1' to enable the entire

interrupt mask. Finally, the whole maskable interrupt system is enabled by means of an EI instruction.

When an interrupt occurs (associated either with a logic '1' being applied to one of the hardware interrupt pins or an RST n instruction being executed), the following operations occur

1 The return address to the main program is pushed onto the stack and the *maskable interrupts are disabled* (these are automatic operations and are not written into the program). The latter means that the interrupt routine cannot itself be interrupted until the interrupts are reenabled.
2 The address of the first instruction in the interrupt routine is moved into the PC. This results in control being transferred to the interrupt handling routine comprising sections 3 through 7 below.
3 The status of the CPU is saved by means of a series of PUSH instructions as described under subroutines.
4 The main body of the interrupt routine is executed.
5 The status of the CPU is restored by means of a series of POP instructions in the manner described under subroutines.
6 The maskable interrupts are enabled by means of a 1-byte EI instruction. This instruction must be included in the program if the interrupt is likely to occur more than once. This makes the interrupt routine reentrant (leaving it out means that the interrupts remain disabled once the interrupt routine has been executed).
7 A return is made to the main program by means of a RETurn instruction.

Sections 3 and 5 are optional and, in some cases, can be omitted. The EI instruction is frequently the penultimate instruction in the interrupt routine because the interrupts *are not enabled until the next instruction in the program has been executed*; that is, the interrupts are not enabled until the RETurn instruction has been executed. This means that the interrupts *are enabled* by the time that the return to the main program has been made.

As with subroutines, the RETurn instruction POPs the return address from the top of the stack into the PC, allowing an orderly return to be made to the main program.

The hardware interrupt INTR does not have a fixed vector address; the vector address for this is placed or is 'jammed' onto the data bus by the interrupting device when the CPU

acknowledges the interrupt. The CPU reads the vector address from the data bus and transfers control to the address in the way described earlier in this section.

Multiple interrupts

If more than one of the maskable interrupts is unmasked, several interrupt requests can occur simultaneously. The programmer must therefore establish the priority of each of the interrupting sources to allow a high priority interrupt to interrupt a low priority routine, but not vice versa. An interrupt priority frequently adopted is as follows

highest	RST 7.5
	RST 6.5
lowest	RST 5.5

The following illustrates the way in which the three interrupts are organized to give the above priority (the priority can, of course, be altered by modifying the program). In the following INTRL is the low priority interrupt (RST 5.5), INTRM is the medium priority interrupt (RST 6.5), and INTRH is the high priority interrupt (RST 7.5).

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A,08H       ;UNMASK RST 7.5, 6.5 AND 5.5
        SIM
        EI              ;ENABLE INTERRUPTS
        ....            ;MAIN PROGRAM COMMENCES
        ....
INTRL:  MVI A,09H       ;UNMASK RST 7.5 AND RST 6.5
        SIM
        EI              ;ENABLE HIGHER PRIORITY
                        ;INTERRUPTS
        ....            ;LOW PRIORITY ROUTINE
                        ;COMMENCES
        MVI A,08H       ;UNMASK ALL INTERRUPTS
        SIM
        RET             ;RETURN FROM INTRL
        ....
        ....
```

```
INTRM:  MVI A,0BH       ;UNMASK RST 7.5
        SIM
        EI              ;ENABLE HIGHEST PRIORITY
                        ;INTERRUPT
        ....            ;MEDIUM PRIORITY INTERRUPT
                        ;COMMENCES
        MVI A,08H       ;UNMASK ALL INTERRUPTS
        SIM
        RET             ;RETURN FROM INTRM
        ....
        ....
INTRH:  XXXX            ;FIRST INSTRUCTION IN
        ....            ;HIGHEST PRIORITY ROUTINE
        ....
        EI              ;ENABLE INTERRUPTS
        RET             ;RETURN FROM INTRH
```

When INTRL occurs, the higher priority interrupts are immediately unmasked and enabled, allowing either of them to interrupt INTRL. Since the interrupts are enabled early in the INTRL routine, it is not necessary to enable the interrupts at the end of the INTRL routine. All the maskable interrupts are unmasked before leaving INTRL so that it is reentrant.

When INTRM occurs, the highest priority interrupt is immediately unmasked and enabled. All interrupts are unmasked before leaving INTRM.

INTRH is a conventional interrupt routine, and all other interrupts remain disabled during its execution. The interrupts are enabled by the EI instruction towards the end of the INTRH routine.

TRAP

TRAP is a nonmaskable interrupt which cannot be masked out or disabled by software, and is both edge and level sensitive. The TRAP interrupt is reserved for the highest priority interrupts such as a power failure situation.

Stack manipulation

The stack pointer can be incremented by means of a 1-byte INX SP instruction, and can be decremented by means of a 1-byte DCX SP instruction. The 16-bit contents of the stack pointer can be exchanged with the contents of the H and L register pair by an XTHL instruction, thus allowing two stacks to be manipulated. Data stored on the stack can be altered by means of POP and PUSH instructions. Additionally, the stack pointer can be loaded from the H and L register pair by means of a SPHL instruction.

Reset

The $\overline{\text{RESET IN}}$ is used to initialize or to reset the microcomputer from a 'power down' situation. When this line of the CPU is taken low, all CPU read/write ceases and all peripherals connected to the RESET OUT line of the CPU are reset. It also sets the contents of the PC to 0000H and resets the Interrupt Enable (IE) flip-flop and the HoLD Acknowledge (HLDA) flip-flop. None of the flags or registers is affected. When the $\overline{\text{RESET IN}}$ line is taken high, control is transferred to the instruction in location 0000H.

A signal on the reset line is often used to retrieve the situation in the event of a system runaway; pressing the reset button transfers control to the instruction in location 0000H.

Serial output and serial input using the SOD and SID lines

The 8085 CPU has on-chip Serial Output Data (SOD) and Serial Input Data (SID) lines which are used in association with the interrupt mask register (*see* **Symbols and abbreviations**).

Serial output The general procedure is as follows.

1 The data bit to be output to the SOD pin is moved into bit 7 of the accumulator.
2 A logic '1' is moved into bit 6 of the accumulator (this is used as a Serial Output Enable (SOE) bit to 'enable' the SOD pin—see **interrupt mask register**).
3 The SOD and SOE bits are moved to the interrupt mask register by means of a SIM instruction.

The following program serializes the data stored in location

2000H (commencing with the l.s.b.). A time delay program (see **Sample programs** section) resides at the symbolic address DELAY; this program is used to generate the correct bit rate or baud rate for serial transmission.

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI C,08H       ;LOAD BIT COUNT INTO REGISTER C
        LDA 2000H       ;GET DATA
        MOV B,A         ;SAVE DATA
NEXT:   MOV A,B         ;GET "SAVED" DATA
        RRC             ;SHIFT L.S.B. INTO SOD POSITION
        MOV B,A         ;SAVE SHIFTED DATA
        ANI 80H         ;LEAVE SOD BIT IN REGISTER A
        ORI 40H         ;SET SOE BIT IN REGISTER A
        SIM             ;OUTPUT DATA TO SOD PIN
        LXI D,D16       ;GENERATE BIT RATE
        CALL DELAY
        DCR C           ;ALL BITS OUTPUT?
        JNZ NEXT        ;NO, GET NEXT BIT
        ....            ;PROGRAM CONTINUES
```

Serial input The contents of the interrupt mask are read by means of a RIM instruction, the serial input bit being in the bit 7 position of the accumulator. The following sequence enables the bit which is applied to the SID pin of the CPU to be stored in a register (register L in this case). Using the sequence as part of an n-stage process, an n-bit serial input word can be converted into a parallel word in register L.

```
GET:    RIM         ;READ SID DATA BIT
        RAL         ;MOVE SERIAL BIT INTO CY FLAG
        MOV A,L     ;GET PARALLEL WORD
        RAL         ;ROTATE SERIAL DATA BIT INTO BIT 0
        MOV L,A     ;SAVE PARALLEL WORD
        ....        ;WAIT FOR NEXT SERIAL DATA BIT.
        ....        ;IF PARALLEL WORD NOT COMPLETE,
        ....        ;JUMP TO "GET" NEW BIT
        ....        ;OTHERWISE CONTINUE
```

Input/Output

There are a large number of I/O devices which are compatible with the 8085 CPU. While it is beyond this Guide either to give details

or to list all these devices, a list of several popular devices is given below. Details are given separately about the 8155 and 8156, which has static RAM with I/O ports and a timer.

(a)	8155 and 8156	Static RAM with I/O ports and Timer.
(b)	8212	8-bit I/O port.
(c)	8251A	Programmable Communications Interface.
(d)	8257	Programmable DMA Controller.
(e)	8272A	Single/Double Density Floppy Disc Controller.
(f)	8275H	Programmable CRT Controller.
(g)	8295	Dot Matrix Printer Controller.
(h)	82062	Winchester Disc Controller.
(i)	82720	Graphics Display Controller.

The manufacturers' data sheets should be consulted for the full specification, pin connections, electrical characteristics and timing diagrams.

8155 and 8156 Static RAM with I/O ports and Timer

The 8155 and 8156 chips (referred to later simply as the 8155) have the following facilities

(a) a 256 (1/4K) × 8-bit static RAM
(b) two 8-bit I/O ports
(c) one 6-bit I/O port
(d) a 14-bit timer
(e) interrupt facilities.

The 8155 and 8156 chips differ only in the respect that the 8155 has an active low chip enable ($\overline{CE}$) and the 8156 has an active high chip enable (CE). Two versions of each chip are available, namely the 'H' version and the 'H-2' version, e.g., the 8155H and the 8155H-2. The 'H' version has a maximum access time of 400 ns, and the maximum access time for the H-2 version is 330 ns. The 6-bit port (port C) can be programmed either as an I/O port or as a status port, the latter mode allowing the two 8-bit ports (port A and port B) to operate in a strobed or handshake mode. The 8155 has the following internal registers.

PA	8-bit I/O port
PB	8-bit I/O port
PC	6-bit I/O or status port
TIMER LOW	Low eight bits of the timer count length
TIMER HIGH	High six bits of the timer count length and two timer mode bits
COMMAND/STATUS	8-bit Command and Status (C/S) registers; both reside at the same address.

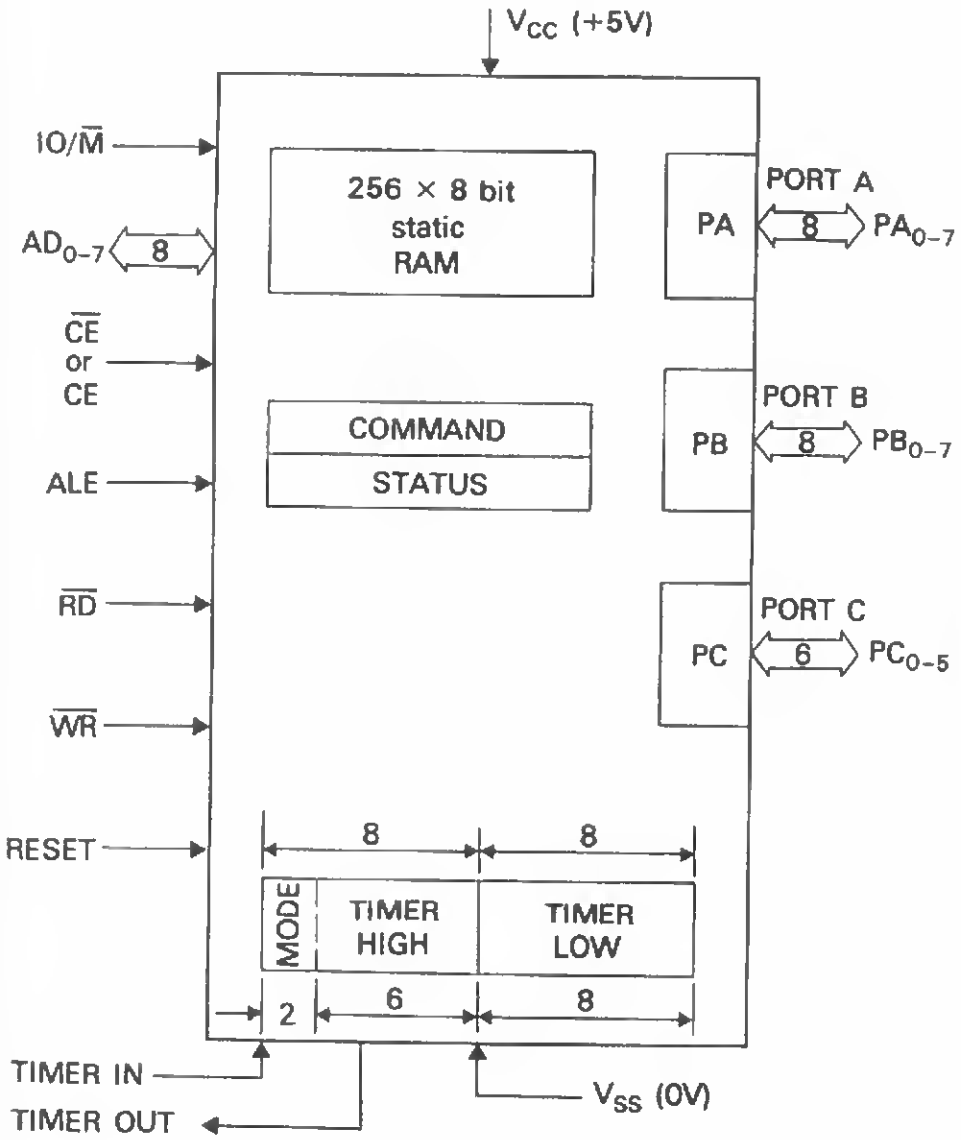


Fig. 4

The functions of the signals connected to the 8155 chip (see Figure 4) are as follows.

$IO/\overline{M}$ selects memory if low or, if high, selects I/O (or C/S) registers.

AD_{0-7} 3-state address/data bus lines. The 8-bit address is latched into the address latch on the falling edge of the ALE signal. The address may either be in the memory or in the I/O section (depending on the state of $IO/\overline{M}$).

$\overline{CE}$ or CE chip enable signal ($\overline{CE}$ on 8155 and CE on 8156).

ALE Address Latch Enable signal from the 8085 CPU; on its falling edge it latches not only the address but also the chip enable signal and the $IO/\overline{M}$ signal into the chip.

$\overline{RD}$ Active low read control line.

$\overline{WR}$ Active low write control line.

TIMER OUT Timer output signal; this can either be a square wave or a pulse, depending on the timer mode.

PA_{0-7} Eight general-purpose I/O lines from port A.

PB_{0-7} Eight general-purpose I/O lines from port B.

PC_{0-6} Six lines from port C which can either function as I/O lines or as the following control lines:

- PC_0 A INTR (port A INTeRrupt)
- PC_1 A BF (port A Buffer Full)
- PC_2 A STB (port A STroBe)
- PC_3 B INTR (port B INTeRrupt)
- PC_4 B BF (port B Buffer Full)
- PC_5 B STB (port B STroBe)

Register address within the 8155

Register	*Binary address*	*Example address (hex)*
Command/Status (C/S)	XXXXX000	20
Port A	XXXXX001	21
Port B	XXXXX010	22
Port C	XXXXX011	23
Timer (low byte)	XXXXX100	24
Timer (high bits and timer mode)	XXXXX101	25

where X = either '1' or '0'

Programming the Command/Status register

Since both the command register and the status register reside at the same address, they are collectively referred to as the **Command/Status (C/S) register**. Data can be *written into the command register* by an OUTput instruction, e.g., OUT 20H, and data can be *read from the status register* by an INput instruction, e.g., IN20H.

The **command register** contains eight latches as shown below.

Bit no.	7	6	5	4	3	2	1	0
	TM_2	TM_1	IEB	IEA	PC_2	PC_1	PB	PA

The function of each of the bits is described below.

Bit No.	*Function*
0	PA: Defines PA_{0-7} } 0 = input, 1 = output
1	PB: Defines PB_{0-7} } 0 = input, 1 = output
2,3	PC: Defines PC_{0-5} as follows (full details given in the section on port C) 00 = ALT 1 11 = ALT 2 01 = ALT 3 10 = ALT 4
4	IEA: '1' to enable port A interrupt, '0' to disable it.
5	IEB: '1' to enable port B interrupt, '0' to disable it.
6	TM_1:TM_2 and TM_1 form a binary pair which define
7	TM_2: the timer count length and its operating mode as follows 00 = NOP—do not affect the counter operation. 01 = STOP (or NOP if counter has not started). 10 = STOP AFTER TC (Terminal Count) is reached (or NOP if counter has not started). 11 = START—load MODE (2 bits) and COUNT LENGTH (14 bits) and start immediately after loading provided that the counter is not running. If the counter is running, start new mode and count length immediately after present TC is reached.

Examples of programming the control (C/S) register
The control register may be configured in one of several ways, the following being examples.

Example 1
```
MVI A,00H    ;TIMER OFF, CHIP INTERRUPTS DISABLED,
OUT 20H      ;PORT C = ALT 1, PORT B = INPUT,
             ;PORT A = INPUT.
```

Example 2
```
MVI A,OFH    ;TIMER OFF, CHIP INTERRUPTS DISABLED,
OUT 20H      ;PORT C = ALT 2, PORT B = OUTPUT,
             ;PORT A = OUTPUT.
```

Example 3
```
MVI A,C1H    ;START TIMER, CHIP INTERRUPTS DISABLED,
OUT 20H      ;PORT C = ALT 1, PORT B = INPUT,
             ;PORT A = OUTPUT.
```

The **status register** contains six latches as shown below.

Bit no.	7	6	5	4	3	2	1	0
	X	TIMER	INTE B	B BF	INTR B	INTE A	A BF	INTR A

The function of each of the bits is described below.

Bit No.	*Function*
0	INTR A: port A interrupt request
1	A BF: port A buffer full/empty
2	INTE A: port A interrupt enable
3	INTR B: port B interrupt request
4	B BF: port B buffer full/empty
5	INTE B: port B interrupt enable
6	TIMER: timer interrupt. This bit is latched high when TC is reached, and is reset low either on reading the C/S register or on starting a new count.
7	not used.

The status of the timer and I/O section can be polled by reading the state of the status register, e.g., by means of an IN XXXXX000B instruction (IN 20H being an example).

Port A (PA) and port B (PB) registers

Each is an 8-bit register which can be programmed to act either as an input port or as an output port (see the programming examples for the C/S register). Additionally, either or both ports can be programmed to operate in a strobed mode, with port C acting as the controlling port.

Port C (PC) register

This is a 6-bit port which can be programmed to act either as an input port, or as an output port, or to provide control signals for PA and PB during strobed (handshake) data transfers. The latter mode is achieved by programming PC to operate either in ALT 3 (ALTernate mode 3) or in ALT 4 (see also the description of the command register above). The function of the pins of port C when used in the strobed mode are listed below.

Pins 0 and 3 — These provide an INteRrupt signal which is generated by the CPU and is *output to* the peripheral.

Pins 1 and 4 — These provide an *output signal* from the CPU to the peripheral to indicate that the Buffer register is Full (or is empty).

Pins 2 and 5 — These accept an *input signal* (the **strobe signal**) from the peripheral.

A table showing all the control assignments of the 8155 is given below.

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	input	output	A INTR	A INTR
PC1	input	output	A $\overline{\text{BF}}$	A $\overline{\text{BF}}$
PC2	input	output	A $\overline{\text{STB}}$	A $\overline{\text{STB}}$
PC3	input	output	output	B INTR
PC4	input	output	output	B $\overline{\text{BF}}$
PC5	input	output	output	B $\overline{\text{STB}}$

Thus, in ALT 1, PC acts as a 6-bit input port; in ALT 2 it acts as a 6-bit output port; in ALT 3, it controls PA in a strobed mode and three of its pins can be used as an output port; in ALT 4 it controls

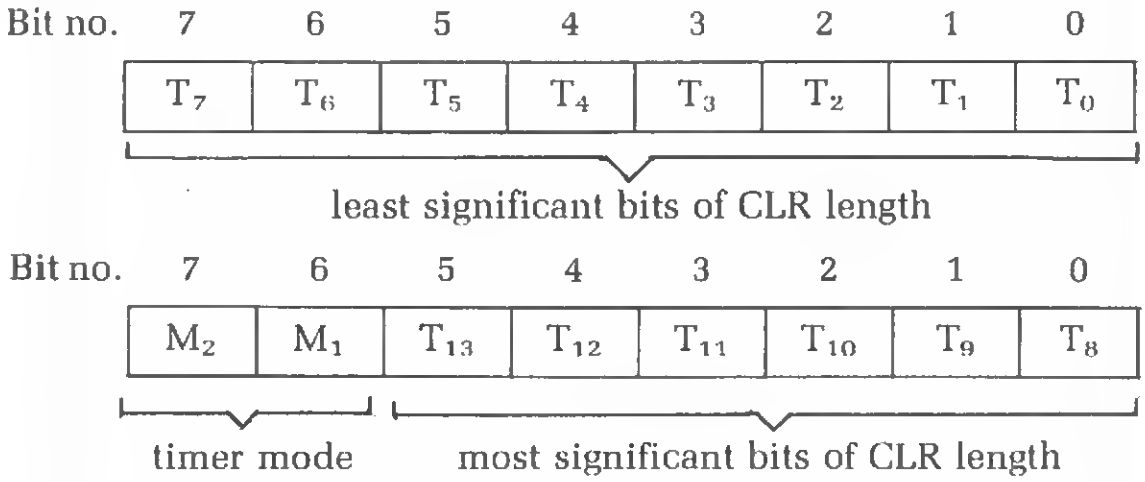
PA and PB in a strobed mode. When PC is programmed either in ALT 3 or ALT 4, the control signals are initialized as follows.

Control function	*Input mode*	*Output mode*
BF	low	low
INTR	low	high

Timer register

This is a 14-bit timer that counts pulses applied to the TIMER INput pin of the chip. The 8155 chip provides either a square wave or a pulse at the TIMER OUT pin when the *terminal count* (TC) is reached.

The timer register is 16 bits in length, and comprises two 8-bit registers (the most significant register performing a dual function). The organization of the two 8-bit registers is shown below.



The least significant byte of the Count Length Register (CLR) is stored in a register at address XXXXX100B (an example of this address being 24H), and the most significant bits of CLR are stored in a register at address XXXXX101, e.g., 25H. The *timer mode bits* M_1 and M_2 are used as follows:

If $M_1 = 1$, the timer operates continuously with an automatic reload at TC.

If $M_1 = 0$, the timer produces a single output (either a pulse or a logic '1' for half the time period).

also

If $M_2 = 1$, the timer produces a short pulse.

If $M_2 = 0$, the timer produces a square wave.

There are therefore four timer modes as follows:

Mode	M_2	M_1	Output
0	0	0	Output low during second half of count
1	0	1	Square wave output (automatic reload)
2	1	0	Single pulse at TC
3	1	1	Pulse each time TC is reached (automatic reload)

In many applications, the CPU clock is connected to the TIMER IN pin, and the TIMER OUT pin is connected to one of the CPU interrupt pins. In this case, the timer mode and period must be loaded and the timer started *before* the interrupts are enabled. The timer can be stopped at any time by software by clearing bit 7 and setting bit 6 of the C/S register.

Programming example using the 8155 programmable timer
Consider a timer which is driven by a clock signal generator having a period of 325 ns, and is to be programmed to produce a pulse every 5 ms. The pulse is used to interrupt an 8085 CPU via its RST 7.5 interrupt pin. The *decimal* count length is calculated as follows:

$$\begin{aligned}\text{decimal count length} &= \text{required pulse period/clock period}\\ &= 5 \times 10^{-3}/(325 \times 10^{-9})\\ &= 15\,385_{10}\end{aligned}$$

hence hex count length = 3C19H

Thus, 19H must be stored in the timer low-order CLR, and 3CH must be stored in bits 0 to 5 of the high-order timer CLR. Moreover, since the timer is to produce repetitive pulses, the mode bits M_2 and M_1 of the high-order timer must both be set to '1'. That is, the high-order timer register must store FCH. The initialization section of the timer program (below) uses the following addresses for the 8155 registers:

C/S register	20H
low-order timer CLR	24H
high-order timer CLR	25H

```
LXI SP,4FFFH    ;INITIALIZE SP
SUB A           ;CLEAR PULSE COUNT STORAGE LOCATION
STA PULSE
MVI A,19H       ;LOAD LOW-ORDER TIMER CLR
OUT 24H
MVI A,FCH       ;LOAD MODE AND HIGH-ORDER TIMER CLR
                ;BITS
OUT 25H
MVI A,C1H       ;INITIALIZE PORT A = OUTPUT AND START
                ;TIMER
OUT 20H
MVI A,0BH       ;UNMASK RST 7.5
SIM
EI              ;ENABLE INTERRUPTS
. . . .         ;MAIN PROGRAM COMMENCES
```

Sample programs

8-bit addition

Hexadecimal addition (ignoring carry-in)
This program adds the hexadecimal data (the addend) in location 2001H to the hexadecimal data (the augend) in location 2000H, and stores the sum in location 2002H.

```
        LXI H,200H    ;LOAD POINTER M
        MOV A,M       ;GET AUGEND
        INX H         ;POINT TO ADDEND
        ADD M         ;ADD ADDEND
        INX H
        MOV M,A       ;STORE SUM
CONT:                 ;CONTINUE
```

Hexadecimal addition (with carry-in)
The ADD M instruction in the previous program is altered to ADC M.

Decimal addition

To add the *decimal* data in location 2000H to the *decimal* data in location 2001H, and to store the sum in location 2002H, the following change is necessary: insert a 1-byte DAA instruction between the ADD M and the second INX H instruction. The decimal addition then occurs *without a carry-in* if an ADD M instruction is used; if an ADC M instruction is used, the carry-in from the CY flag is also added.

Multi-byte addition

This program adds a number of bytes of hexadecimal data in a table commencing in location 2001H (the least significant byte being in location 2001H) to the same number of bytes in a table commencing in location 2010H, the sum being stored in successive locations from locations 2001H onwards (least significant byte in location 2001H). The length of the table (in hex) is given by the value in location 2000H.

```
        LXI H,2000H   ;LOAD POINTER M
        MOV B,M       ;SAVE TABLE LENGTH IN REGISTER B
        INX H         ;POINT TO ADDEND BYTE
        LXI D,2010H   ;LOAD AUGEND POINTER
        ANA A         ;CLEAR CARRY FLAG FOR FIRST
                      ;ADDITION
LOOP:   LDAX D        ;LOAD AUGEND INTO ACCUMULATOR
        ADC M         ;ADD ADDEND (WITH CARRY)
        MOV M,A       ;MOVE SUM INTO M
        INX D         ; INCREMENT AUGEND POINTER
        INX H         ;INCREMENT ADDEND POINTER
        DCR B         ;ALL DATA ADDED?
        JNZ LOOP      ;NO, ADD NEXT BYTE
CONT:                 ;OTHERWISE CONTINUE
```

The program can be modified to handle decimal addition (assuming that the data is stored in decimal form) by including a DAA instruction *after* the ADC M instruction.

8-bit subtraction

This program subtracts the hexadecimal data (the subtrahend) in location 2001H from the hexadecimal data (the minuend) in location 2000H, and saves the difference in location 2002H.

```
        LXI H,2000H   ;LOAD POINTER M
        MOV A,M       ;LOAD MINUEND INTO ACCUMULATOR
        INX H         ;POINT TO SUBTRAHEND
        SUB M         ;AND SUBTRACT IT
        INX H
        MOV M,A       ;SAVE DIFFERENCE
CONT:                 ;CONTINUE
```

The above program ignores any borrow-in; the borrow-in may be accounted for if the SUB M instruction is replaced by a SBB M instruction.

8-bit multiplication

This program multiplies the hexadecimal value (the multiplicand) in location 2000H by the hexadecimal value (the multiplier) in location 2001H, the low byte of the 16-bit product being stored in location 2002H and the high byte in location 2003H.

In the program, the register pair D and E is used to store the multiplicand (register D contains zero [00H] and register E stores the 8-bit multiplicand), and the register pair H and L store the partial product during the calculation. The following terms are used in the program

MD = multiplicand
MR = multiplier
PP = partial product

```
        LXI H,2000H     ;LOAD POINTER M
        MOV E,M         ;GET MD AND EXTEND ITS
        MVI D,00H       ;LENGTH TO 16 BITS
        INX H           ;POINT TO MR
        MOV A,M         ;GET MR
        LXI H,0000H     ;CLEAR PP REGISTER
        MVI B,08H       ;LOAD BIT COUNT
MULT:   DAD H           ;SHIFT PP LEFT
        RAL             ;ROTATE MULTIPLICATION BIT
                        ;INTO CY FLAG
        JNC NOCARRY     ;JUMP TO NOCARRY IF CY = 0
        DAD D           ;OTHERWISE PP = PP + MD
NOCARRY: DCR B          ;MULTIPLICATION COMPLETE?
        JNZ MULT        ;NO, DEAL WITH NEXT BIT
        SHLD 2002H      ;STORE PRODUCT
CONT:                   ;CONTINUE
```

Moving a block of data

This program moves a block of data commencing at an address whose low byte is stored in location 2000H and whose high byte is stored in address 2001H, to a new block commencing at an address whose low byte is stored in location 2002H and whose high byte is stored in location 2003H. The number of bytes to be moved (in hex) is stored in location 2004H. The register pair H and L are used to store the base address of the data source, and the register pair D and E store the base address of the destination.

```
        LDA 2000H   ;LOAD H AND L REGISTER PAIR WITH
        MOV L,A     ;BASE ADDRESS OF DATA SOURCE
        LDA 2001H
        MOV H,A
        LDA 2002H   ;LOAD D AND E REGISTER PAIR WITH
        MOV E,A     ;BASE ADDRESS OF DATA DESTINATION
        LDA 2003H
        MOV D,A
        LDA 2004H   ;LOAD REGISTER C WITH THE NUMBER
        MOV C,A     ;OF BYTES TO BE MOVED
LOOP:   MOV A,M     ;GET DATA BYTE FROM SOURCE
        STAX D      ;ADDRESS AND STORE IT IN
                    ;DESTINATION ADDRESS
        INX H       ;INCREMENT SOURCE ADDRESS
        INX D       ;INCREMENT DESTINATION ADDRESS
        DCR C       ;ALL DATA MOVED?
        JNZ LOOP    ;NO, GET NEW BYTE OF DATA
CONT:               ;CONTINUE
```

Software time delay

A software time delay is generated by means of a time-wasting program which simply uses up a known number of clock cycles while not performing a useful function. The general procedure is as follows:

1 load a register (or register pair) with a specified value.
2 decrement the register contents until they reach zero.

To generate a longer time delay, steps 1 and 2 are repeated several times. The following program uses the register pair D and E to

store a 16-bit value which is decremented on each pass of a timing loop. The value (the data) stored in the register pair is calculated as follows

$$\textit{Decimal}\text{ value of the data} = \frac{\text{number of clock pulses used} \times \text{clock period of CPU}}{\text{required time delay}}$$

The program uses the following instructions:

Instruction	Clock cycles
LXI D,D16	16
DCX D	4
MOV A,D	4
ORA E	4
JNZ Adr	7/10
RET	10

The JNZ Adr instruction uses 10 clock cycles when a jump occurs, and 7 clock cycles when no jump occurs. The program is written in the form of a subroutine, and is terminated with a RET instruction. An example of its use is given in the program in the **Control of I/O port with flashing lights** section. The method used to calculate the value of the data (D16) loaded in the register pair D and E is illustrated later in this section.

```
TIME:   LXI D,D16   ;LOAD TIME DELAY DATA
DELAY:  DCX D       ;DECREMENT REGISTER PAIR D AND E
        MOV A,D     ;(A) ← (D)
        ORA E       ;LOGICALLY OR (E) WITH (D)
        JNZ DELAY   ;JUMP IF DELAY INCOMPLETE
        RET         ;OTHERWISE RETURN TO CALLING
                    ;PROGRAM
```

The DELAY loop from the instruction DCX D to JNZ DELAY is executed (data [D16] − 1) times, and the number of clock cycles used by the program is

$$16 + ([6+4+4+10] \times [\text{data} - 1]) + (6+4+4+7+10)$$

The first 16 clock cycles are for the LXI D,D16 instruction. The [6+4+4+10] clock cycles are for the group of instructions DCX, MOV, ORA and JNZ *with a jump*; the (6+4+4+7+10) clock cycles are for the group of instructions DCX, MOV, ORA, JNZ *without a*

jump, and RET on the final pass of the loop. That is, the total number of clock cycles needed is

$$47 + (24 \times [\text{data} - 1])$$

If, for example, a time delay of 200 ms is required and the clock period of the CPU is 500 ns, then

$$200\ \text{ms} = \{47 + (24 \times [\text{data} - 1])\} \times 500\ \text{ns}$$

hence $\text{data} = 16666_{10}$ or 411AH

Thus, the first instruction in the time delay program is LXI D,411AH. The reader should note that the DCX D instruction *does not set the zero flag* (see details of the arithmetic group of the 8085 instruction set classification). Since it is necessary to determine when the contents of the register pair D and E reach zero, the MOV A,D and the ORA E instructions are included in the program; the function of the latter instruction is either to set or to clear the zero flag prior to its being tested.

Control of an I/O port

Eight LEDs are connected to port A of an 8155 programmable I/O port and eight switches are connected to port B. The following program allows the LED connected to line A_n of port A to be controlled by the switch connected to line B_n of port B.

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A,01H       ;PORT A = OUTPUT, PORT B = INPUT
        OUT CSR
GET:    IN PORTB        ;READ STATE OF SWITCHES AND
        OUT PORTA       ;OUTPUT IT TO LEDs
        JMP GET         ;GET NEW DATA
```

Control of an I/O port with flashing LEDs

The LEDs in the I/O port program above can be made to flash on and off at a regular rate when the switches are in the logic '1' position by including a time delay routine as follows (see also the section on the **Software time delay**).

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A,01H       ;INITIALIZE PORT
        OUT CSR
GET:    IN PORTB        ;READ STATE OF SWITCHES AND
        OUT PORTA       ;OUTPUT IT TO LEDs
        CALL TIME       ;MAINTAIN STATE OF LEDs FOR DELAY
                        ;PERIOD
        SUB A           ;CLEAR ACCUMULATOR AND
        OUT PORTA       ;EXTINGUISH LEDs
        CALL TIME       ;MAINTAIN LEDs EXTINGUISHED FOR
                        ;DELAY PERIOD
        JMP GET         ;GET NEW DATA
```

Bit testing

There are three general methods used to test the value of a bit in an 8085 microcomputer word as follows:

1 The state of bit b_7 can be tested by means of either a JP instruction (for bit $b_7 = 0$) or JM instruction (for bit $b_7 = 1$). Note: it is first necessary to ensure that the instruction before the conditional jump instruction alters the sign flag.
2 Rotate instructions (RAL, RAR, RRC or RLC) can be used to transfer the required bit into the CY flag. The flag can then be tested either by a JC instruction (CY=1) or a JNC instruction (CY = 0).
3 A logical AND bit mask can be used to leave the state of the required bit in the accumulator. For example, ANI 08H leaves the state of bit 3 alone and causes all other bits to be zero; this is followed either by a JZ instruction (jump if the bit being tested = 0) or a JNZ instruction (jump if the bit being tested = 1).

Digital-to-Analog Conversion (DAC) applications

The following programs generate waveforms by outputting data to a DAC connected to port A of an 8155 chip. The output polarity from the DAC depends on the type of binary code employed by the DAC (which may be, for example, either pure binary, or sign-and-modulus, or two's complement, or biased (offset) binary); in the following, the CPU outputs a pure binary code to the DAC.

Rectangular waveform

This program generates a rectangular waveform with a 1:1 mark-to-space ratio; it is possible to alter not only the mark-to-space ratio but also the frequency by altering the time delay parameter used in each half of the waveform. The amplitude of the wave can be altered by changing the data associated with the two MVI A,D8 instructions.

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A.01H       ;PORT A = OUTPUT
        OUT CSR
MAX:    MVI A,FFH       ;OUTPUT MAXIMUM VALUE TO DAC
        OUT PORTA
        CALL TIME       ;TIME DELAY TO ESTABLISH "MARK"
                        ;PERIOD
        MVI A,00H       ;OUTPUT MINIMUM VALUE TO DAC
        OUT PORTA
        CALL TIME       ;TIME DELAY TO ESTABLISH "SPACE"
                        ;PERIOD
        JMP MAX         ;REPEAT CYCLE
```

Rising ramp waveform

This program produces an output waveform which rises linearly with time to a maximum value, after which it falls to zero and begins to rise again. The minimum value of the wave can be made nonzero by starting the binary value from a nonzero value; the maximum value can be altered by terminating the ramp at some other value than FFH. The frequency of the wave can be altered by changing the time delay parameter associated with the TIME subroutine.

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A,01H       ;PORT A = OUTPUT
        OUT CSR
        SUB A           ;CLEAR ACCUMULATOR
        MOV B,A         ;SAVE RAMP DATA
RAMP:   OUT PORTA       ;OUTPUT RAMP DATA TO DAC
        CALL TIME       ;FREQUENCY CONTROL SUBROUTINE
        INR B           ;INCREMENT RAMP DATA
        MOV A,B         ;GET NEW RAMP DATA
        JMP RAMP        ;AND OUTPUT IT
```

Generating a waveform from data stored in a table in memory
This program accesses a table of hexadecimal data stored in sequential memory locations (commencing at the symbolic address TABLE). The LENGTH of the table is given by a hexadecimal value stored in register B (the length being up to FFH); the data is output to a DAC connected to port A of an 8155 chip. The table is repeatedly scanned to give a repetitive waveform; the frequency of the waveform can be altered by changing the time delay associated with the TIME subroutine.

```
        LXI SP,4FFFH    ;INITIALIZE SP
        MVI A,01H       ;PORT A = OUTPUT
        OUT CSR
START:  LXI H,TABLE     ;LOAD BASE ADDRESS OF TABLE
        MVI B,LENGTH    ;LOAD TABLE LENGTH (IN HEX)
WAVE:   MOV A,M         ;GET DATA FROM TABLE AND
        OUT PORTA       ;OUTPUT IT TO DAC
        CALL TIME       ;FREQUENCY CONTROL SUBROUTINE
        INX H           ;INCREMENT TABLE POINTER
        DCR B           ;DECREMENT TABLE LENGTH
        JNZ WAVE        ;JUMP IF TABLE NOT COMPLETELY
                        ;SCANNED
        JMP START       ;OTHERWISE REPEAT FROM START
                        ;OF TABLE
```